A Preview of
Active Server Pages +

Richard Anderson
Alex Homer
Rob Howard
Dave Sussman

Wrox Press Ltd. ®

A Preview of Active Server Pages +

Published by Wrox Press Ltd
Arden House, 1102 Warwick Road, Acock's Green, Birmingham B27 6BH, UK
Printed in USA
ISBN 1-861004-75-3

Table of Contents

Table of Contents

Foreword

The Story of ASP+

Way back in late November, 1997, a small group of us from the Internet Information Server team at Microsoft started to discuss some new ways of creating web applications. Active Server Pages was really only about a year old at that point, and was becoming an incredibly popular technology. However, we had *just* sent IIS 4.0 and the NT Option Pack off to manufacturing, it was the start of the holiday season, and so things were pretty quiet at the office as people recuperated after the long grind. Therefore, taking advantage of this fact, Scott Guthrie and I started to debate endlessly what we liked and didn't like about ASP. While we loved the ease with which you could develop apps with ASP and the flexibility that it provided, there were a number of issues that we identified.

One of the first things that we decided was that ASP was too complicated, and that you had to write too much code. When we would tell people this, they often couldn't understand what we were talking about. "What do you mean? ASP isn't complicated, it's incredibly simple to use!" They were right – ASP is simple. However, while ASP code is easy to write, it tends to be very unstructured and gets really messy. We had seen some huge ASP pages that had tons of server script code intermixed with client script code and after a while it became mind-bogglingly difficult to figure out what was going on. So yes, the code was simple and straightforward, but there was always a ton of it!

Additionally, to do *anything* in ASP you always had to write code – there was no purely *declarative* method for doing anything. If you wrote an ad rotator component, for example, there was no way to just plop it down on the page and have it do anything. You always had to write some code. The reason was that ASP had no actual component model. Actually having a component model was made more difficult by the fact that an ASP page really defined a single function. You always started at the top of the page, and zipped right down to the bottom, executing your database queries, running your business logic, and generating HTML along the way. Therefore, *where* a component is placed in HTML determines *when* it gets to do its work.

- **Separation of code and content**. In many environments, developing a Web application utilizes the skills of a wide range of professionals – for example, you have programmers writing the code, and designers making the HMTL look good. Having both the code and the content intermixed in a single file that both of these groups need to operate on makes it difficult for them to work together. ASP+ allows code and content to be separated.
- **Simplified deployment**. There are difficulties encountered today with deploying components, where they have to be registered and they are locked by the operating system when in use. Configuration also has to be done through special applications. Both these factors make it hard to deploy, move or replicate your application as a single entity.
- **Support for multiple client types**. It's hard to create pages that work well with all the different types of clients that are using the Web. ASP+ provides rich server-side components that can automatically produce output specifically targeted at each type of client.
- **Support for 'Next Generation' Web Services**. ASP+ provides a new 'Web Services' feature that allows services to be created with native support for consuming and generating XML.
- **Improved availability**. ASP+ is more tolerant of application failures, memory leaks, etc. It can pro-actively restart processes that have failed to improve reliability and availability.
- **Improved scalability**. New session-state features make it easy to create applications that work well in Web farms (with multiple servers) and 'Web gardens' (single servers with multiple processors).
- **Improved tool support**. The rich, component based, event driven programming model that ASP+ implements makes it easier for tools from Microsoft (such as Visual Studio) and from other vendors to be integrated. This makes the development process much more like creating traditional applications, and the development environment can interact better to make the job even easier.

Many of these changes came about because the architecture of ASP+ is now much more modularized. Every page becomes a programmatically accessible fully compiled runtime object, and takes advantage of techniques like object-oriented design, just-in-time compilation, runtime support and dynamic caching. At the same time, the backward-compatible nature of ASP+ means that existing pages and applications can often be used as they stand, and the new features of ASP+ added to them over time. Alternatively, you can run ASP+ pages side by side with your existing ASP pages and migrate to ASP+ at your own pace.

A New Runtime Environment

In Chapter 1, we overview the features of ASP+ and describe the new structure and concepts that it involves. However, ASP+ is only a part of a larger Windows system service. It depends on a new runtime system called the **Next Generation Web Services Runtime** (NGWS Runtime). The runtime provides a completely new, highly efficient and scalable execution environment for ASP+ and for other applications and services.

This new runtime is the foundation on which ASP+ is built, and it provides all the capabilities needed to build the new generation of Web-based distributed applications and dynamic Web sites. The tight integration does mean, however, that you need to understand a bit more about what goes on under the hood to get the best from ASP+. Again, in Chapter 1, we'll look at the main issues that this involves.

What Does This Book Cover?

This book was written as Microsoft was about to release the first preview version of ASP+. This release is almost feature complete, and stable enough for developers to begin learning about and using the new technology. While we can't guarantee that the final release version will be identical, you can be sure that almost all of the concepts, examples and explanations we provide are accurate within the timeframe of the first full version of ASP+.

In this book, we attempt to explain just what ASP+ is all about, how you can use it, and what you can use it for. We start in **Chapter 1** with a look at the overall concepts of ASP+, why and how it has changed, and the things you should know to use it effectively. We also look briefly at the NGWS runtime framework, and see how this integrates with the ASP+ code we write.

Chapter 2 then goes on to look at how we can create dynamic pages with ASP+ in much the same way as we've done with ASP. However, the new architecture of ASP+ allows us to do a lot more than before with a lot less code – the programmer's dream. We show you how ASP+ allows objects that represent HTML controls and elements to be executed on the server, with all the supporting HTML and code required being generated automatically.

To extend the functionality of these server-based controls, we can also use templates and server-side data binding to modify the code that is created. In **Chapter 3**, we show how to add these features to your pages. ASP+ will ship with a new data access technology called ADO+, which is integrated into the data binding process to provide truly disconnected and versatile recordset capabilities.

In **Chapter 4**, we introduce and describe a range of advanced ASP+ programming techniques. This includes the new input validation features, reusable 'pagelet' components, output caching support, and techniques for handling errors and debugging your pages.

Chapter 5 introduces another brand new technique called ASP+ Web Services. It demonstrates how you can create XML Services that are accessible across the Internet. Together with a range of new server-side components that come with ASP+, these allow powerful business-to-business applications to be built without the need for the cumbersome techniques we've used in the past.

In **Chapter 6**, we move on to take a more in-depth technical look at the ASP+ Application Framework. To get the most from ASP+, you need to understand a little more about how requests and responses are managed, and how ASP+ application configuration differs from existing techniques. The good news is that it's much easier than ever before, through a far more flexible and extensible management system.

While the server-side controls provided with ASP+ are undoubtedly powerful and useful, you may need to create your own at some stage, or extend the controls that are provided. This might be in order to achieve new functionality, or to provide broader coverage for different kinds of client-side user agent. **Chapter 7** looks at how we can build our own server-side controls, and add them to the ASP+ environment.

Finally, in **Chapter 8**, we look at an application that was designed and built by Microsoft as an example of the way that ASP+ can be used. It takes advantage of many of the new features in ASP+ and the universal runtime, and will help you to get a feel for how complete applications can be created.

Who Is This Book For?

This book covers a product that is still under development, and as such it is aimed at experienced ASP developers who are working at the leading edge – rather than the casual ASP developer or beginner. For example, we do not cover the basics of COM, ASP, or the programming and scripting languages we use in this book.

However, the fact that (at the time of writing) the product is still a preview version does **not** mean that you can ignore it. ASP+ is going to arrive with the new NGWS runtime package, and it will probably also be part of the next release of Visual Studio (which should contain tools to help you work with ASP+).

So, our aim is to cover conceptual overviews of the product – including some of the background theory and explanation of why the product has developed along the lines it has. This is followed by deeper investigation of the features that developers will use first. We show how to take advantage of the new features quickly and with the minimum of fuss.

Providing that you have used ASP before, and are reasonably comfortable with the concepts, you should be able to use this book without requiring any other reference material (other than the SDK and Help files provided with the product). You should also be comfortable with the general principles of using COM to instantiate and use components, and the Visual Basic (or VBScript) languages. Some of the samples are written in other languages, such as C++, JScript and C# (read as "C Sharp") that are supported by the universal runtime framework, but you don't need to be fluent in these languages to be able to use this book.

Acknowledgements

While we depend on the software manufacturers to help us out with technical support and information for almost all the books we write, we must acknowledge the special situation within which this book was produced. Wrox has been at the forefront of ASP publishing since the first beginnings of this technology, and we are grateful for the regular support we receive from the developers and product managers at Microsoft.

This book, however, began life during a very early stage of the development of ASP+ and has provided us with a unique insight into the development of the product. It certainly could not have been written without the generous and timely assistance of many members of the ASP and COM+ teams at Microsoft, especially Mark Anders, Scott Guthrie, Brad Merrill, Brad Millington, Eric Andrae and the rest of the ASP+ team.

To all of you, thanks guys – we hope you like the result.

What You Need to Use this Book

To run all of the examples, you'll need to have the following software:

- Windows 2000
- Internet Explorer 5.5
- NGWS Runtime
- SQL Server 7

The complete source code for this book is available for download from our website, at http://www.wrox.com. Updates to the book will be available at http://www.wrox.com/beta

Conventions

To help you get the most from the text and keep track of what's happening, we've used a number of conventions throughout the book.

For instance:

> **These boxes hold important, not-to-be forgotten information that is directly relevant to the surrounding text.**

While the background style is used for asides to the current discussion.

As for styles in the text:

- When we introduce them, we **highlight** important words
- We show keyboard strokes like this: *Ctrl-A*
- We show filenames and code within the text like so: `doGet()`
- Text on user interfaces (and URLs) is shown as: Menu

We present code in two different ways:

```
In our code examples, the code foreground style shows new, important,
    pertinent code
while code background shows code that's less important in the present context,
    or which has been seen before.
```

Tell Us What You Think

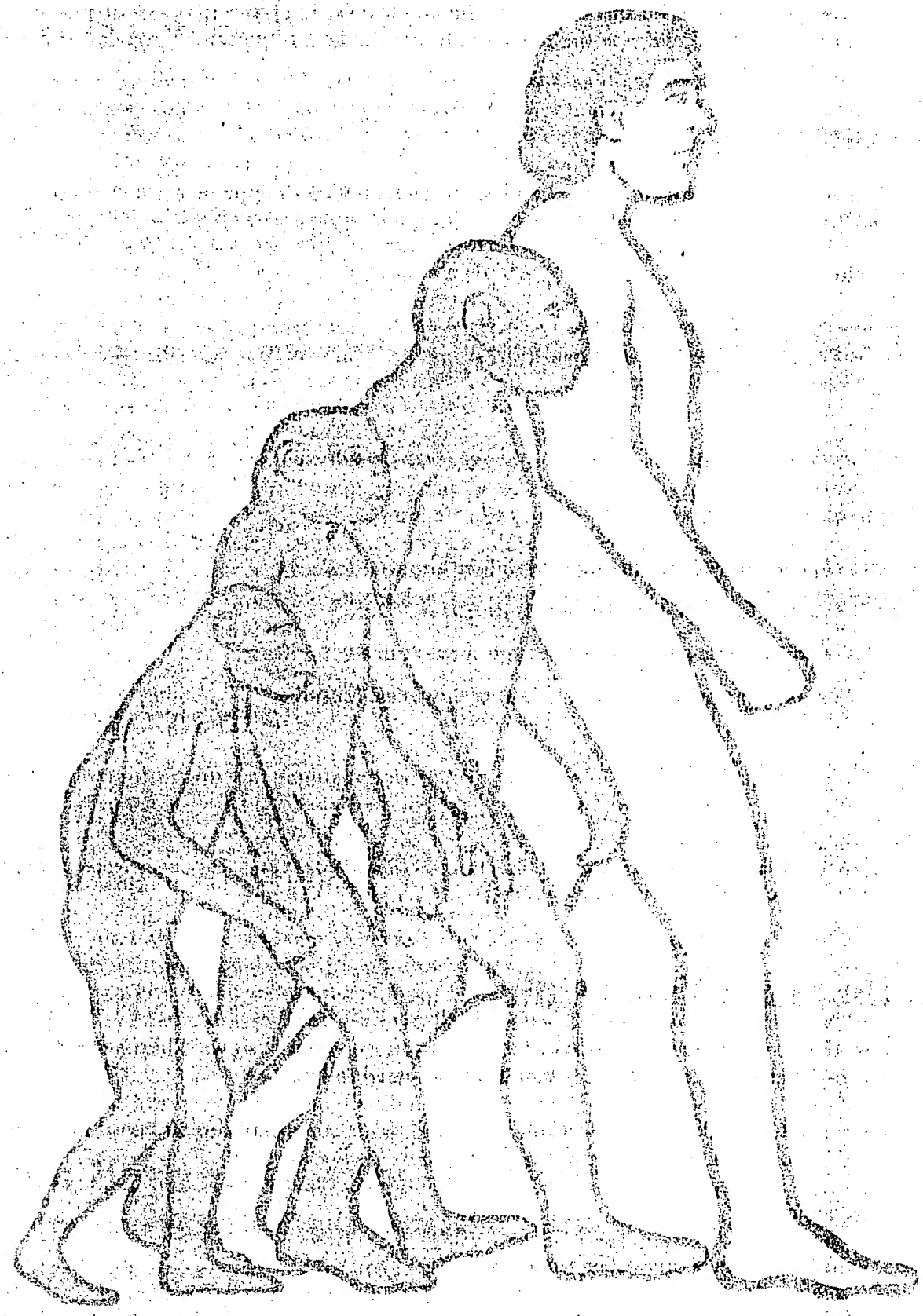
We've worked hard to make this book as useful to you as possible, so we'd like to know what you think. We're always keen to know what it is you want and need to know.

We appreciate feedback on our efforts and take both criticism and praise on board in our future editorial efforts. If you've anything to say, let us know via e-mail:

feedback@wrox.com

or via the feedback links on our web site:

http://www.wrox.com



1

Introducing ASP+

Even though the ink is barely dry on the documentation for Active Server Pages 3.0, Microsoft is already hard at work on the next generation of their core server-side programming technology. In this chapter, we introduce this new product, and look at what it is all about. Currently called **ASP+ Next Generation Web Services** (though this name might yet change) we'll see why we need a new version of ASP, and explore the concepts behind its design and implementation. While this book is aimed predominantly at experienced developers who have used ASP before, we start out in this chapter by examining some of the core issues involved when you decide to migrate to ASP+.

ASP+ is designed to be backwards-compatible with earlier versions of ASP, with only minor changes required in some circumstances (we explore these further in the appendices). However, more to the point, you can install ASP+ on an existing Windows 2000 server alongside ASP 3.0. This allows you to experiment with the new version, without requiring a separate 'test bed' server. You can continue using existing ASP applications, and migrate them ASP+ when you are ready, so your investment in ASP is not lost.

But simply porting your applications to ASP+ will only give you a few of the benefits the new version offers. ASP+ has many new features that provide far greater ease of use, more power and better runtime efficiency, but to take advantage of them you will need to understand more about the way that ASP+ works.

As we are writing this book using a preview version of ASP+, we can't be exactly sure of all the features of the final release. But thanks to the information and assistance provided by the ASP+ team at Microsoft, we can be pretty sure that the content of the book will be reliable and useful with the final version. We'll also be maintaining a special Web site that is accessible from http://www.wrox.com/beta, where we'll document changes as the beta and final release versions appear, and provide some detailed information as well.

So, in this first chapter, we'll cover:

- How Active Server Pages has evolved since its inception
- What the new runtime framework is
- How ASP+ is different to ASP, and why
- A brief guide to getting started with ASP+
- Some of the changes expected in the final release version

We start with a look at the way that ASP and ASP+ have evolved, as this will help to set the background for understanding and working with the new product. For more information about working with COM+ and previous versions of ASP, check out Professional ASP 3.0, (ISBN 1-861002-61-0) from Wrox.

The Evolution of Active Server Pages

Although it seems to have been around forever, **Active Server Pages** is only some three-and-a-bit years old. Since its inception in late 1996, it has grown rapidly to become the major technique for server-side Web programming in the Windows environment (and on some other platforms using other implementations that accept the same or similar syntax, such as ChilliASP). But it didn't come from nowhere – the foundations lie much further back than that.

Dynamic Server-side Web Programming

Traditionally, dynamic Web pages have been created using server-side executable programs. A standardized Web server interface specification called the **Common Gateway Interface** (CGI) allows an executable program to access all the information within incoming requests from clients. The program can then generate all the output required to make up the return page (the HTML, script code, text, etc.), and send it back to the client via the Web server.

To make the programmer's life easier, and save having to create executable programs, languages such as Perl use an application that accepts text-based script files. The programmer simply writes the script, and the Web server executes it using a Perl interpreter.

Microsoft ISAPI Technologies

Microsoft introduced another Web server interface with their Web server, Internet Information Server. This is the **Internet Server Application Programming Interface** (ISAPI), and differs from the CGI in that it allows compiled code within a dynamic link library (DLL) to be executed directly by the Web server. As with the CGI, the code can access all the information in the client request, and it generates the entire output for the returned page.

Most developments in Microsoft's Web arena have been based on the ISAPI interface. One early and short-lived product was **dbWeb**, a data access technology that provided a range of searching, filtering and formatting capabilities for accessing data stored on the server, and for interacting with the client.

A second development was the **Internet Database Connector** (IDC). This proved a big hit with developers – not only because it was fast and efficient (unlike dbWeb), but also because it was a lot more generic and easier to program. IDC introduced the concept of **templates**, allowing programmers to easily adapt existing HTML pages to use its features and quickly build new applications around it.

IDC uses two text files for each 'page'. The first is a simple script that defines the way that the data should be collected from the server-based database. In essence, it is just a SQL statement plus some configuration information:

```
{this is the query file named getuserlist.idc}
Datasource: GlobalExampleData
Username: examples
Password: secret
Template: getuserlist.htx
SQLStatement:
+ SELECT DISTINCT UserName
+ FROM Person ORDER BY UserName;
```

The server executes this file to obtain the results recordset, then loads a template file:

```
{this is an extract from the template file named getuserlist.htx}
...
<TABLE>
  <TR>
    <TD>User name:</TD>
    <TD>
      <SELECT NAME=selUserName>
        <%BeginDetail%>
         <OPTION VALUE="<%UserName%>"><%UserName%>
        <%EndDetail%>
      </SELECT>
    </TD>
  </TR>
</TABLE>
...
```

The template is just an ordinary Web page, including HTML, text and other objects, but with one or more specially delimited placeholders inserted. And the syntax for these placeholders, and the other simple program code constructs that are supported, is eerily like ASP. Of course, it was from this that ASP actually evolved:

The Versions of ASP

So, it was in early 1996 that **Denali** (the codename for ASP) was released as a beta version 0.9 product, and it took the Web-development world by storm. The ability to execute code inline within a Web page was so simple and yet so powerful. With the provision of a series of components that could perform advanced features, most notably **ActiveX Data Objects** (ADO), it was almost child's play to create all kinds of dynamic pages.

The final release version of Active Server Pages 1.0, available as an add-on for IIS 3.0, was soon in use on Windows platforms all over the place. The combination of ASP with ADO enabled developers to easily create and open recordsets from a database. There's no doubt that this was one of the main factors for its rapid acceptance, because now you could create and open recordsets from a database within the script, and manipulate and output any values, in any order, almost any way you wanted.

In 1998, Microsoft introduced ASP 2.0 as part of the free **Windows NT4 Option Pack**. The major difference between this release of ASP and version 1.0 was in the way that external components could be instantiated. With ASP 2.0 and IIS 4.0, it is possible to create an ASP **application**, and within it run components in their own separate memory space (i.e. out of process). The provision of **Microsoft Transaction Server** (MTS) also made it easy to build components that can partake in transactions.

Windows 2000, COM+, and ASP 3.0

Early this year (2000), Windows 2000 arrived. This contains version 5.0 of IIS, and version 3.0 of ASP. Other than some minor additions to ASP, the core difference here is actually more to do with COM+. In Windows 2000, Microsoft combined MTS with the core COM runtime to create **COM+**. This provides a host of new features that make the use of components easier, as well as giving a much more stable, scalable and efficient execution platform.

Other than a few minor changes to the management interface, IIS has not changed a great deal on the surface. However, underneath, it now uses COM+ Component Services to provide a better environment for components to be executed within, including out of process execution as the default and the option to run each component in its own isolated process if required.

ASP+ and the Next Generation Web Services Framework

All this brings us to the present, with ASP+. The underlying structure of ASP+ is very different to that of previous versions, although from the 'outside' (as far as the developer is concerned) it does appear to offer a very similar interface. ASP+ is almost entirely component-based and modularized, and every page, object, and HTML element you use can be a runtime component object.

For this to perform efficiently, and provide a scalable solution, the management of these objects is a very necessary prerequisite. The new runtime environment carries out this management automatically, allowing ASP+ to become far more object-oriented in nature. This lets developers build more powerful applications by accessing these component objects in a far more granular and controlled manner.

On top of that, the object orientation of ASP+ provides extensibility for the environment as a whole. Developers can add to and extend the environment, both by creating new components or inheriting from the base classes that create them, and by over-riding selected behavior as required. Under the hood, the COM+ runtime manages the instantiation, pooling, and allocation of the objects automatically.

The Next Generation Web Services Framework

So, COM+ provides a framework of operating system services. But that's not the whole story. ASP+ is actually a part of a brand new **runtime framework** that provides support for all kinds of applications in Windows. The framework is a key part of of Microsoft's **Next Generation Web Services** or **NGWS**. When you install this, you get ASP+ as part of the package. The NGWS framework supports all other server-side programming techniques as well, such as a new managed component service, support for building executable applications and Windows Services, access to performance counter APIs and Event Log APIs, etc.

The NGWS framework extends the Component Object Model (COM) architecture that we use to create re-usable and interoperable software components by adding new and enhanced services for scalable distributed applications:

- A unified, rich set of programming libraries
- A secure, multi-language runtime engine
- Simplified application creation, deployment and maintenance
- Increased scalability for distributed applications
- Protection of existing software and training investments

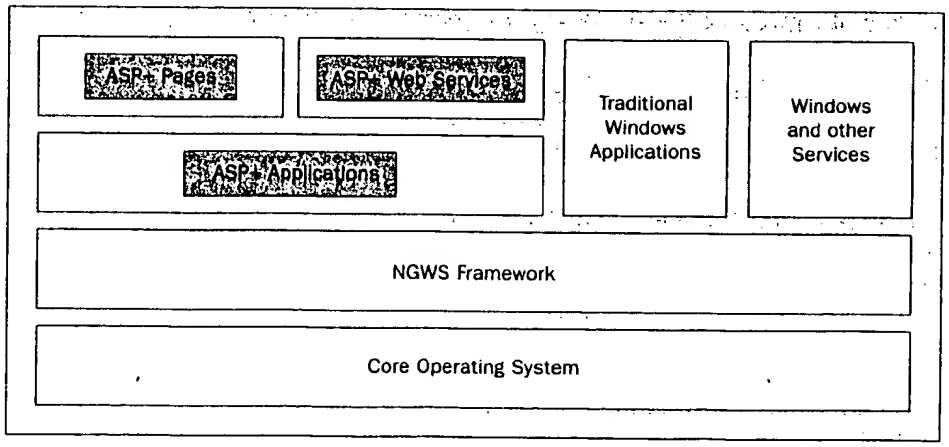
We'll look at how it does all these things next.

What Is the NGWS Framework?

The integration of ASP into the operating system differs remarkably from earlier versions of ASP, which were basically just add-ons to the operating system. Up until now, ASP has been implemented through an ISAPI DLL named `asp.dll`, plus a few new system files and the ASP user components that came as part of the package (such as the Browser Capabilities component).

The NGWS framework reflects the information technology industry's changing view of the needs for creating, deploying, and maintaining Web services of all types – ranging from simple client applications to the most complex distributed architectures. The overall concept and strategy is part of the Windows **Distributed Internet Applications** (DNA) architecture.

However, the important part to recognize is that the framework is **not** just there for ASP+. It acts as a base for all kinds of applications to be built on Windows. The following diagram shows how the runtime framework supports **ASP+ Applications**:



The NGWS framework provides an execution engine to run code, and a family of object oriented classes/components that can be used to build applications. It also acts as an interface between applications and the core operating system. You might ask why we need such a layer, when existing applications can talk to the core operating system and services quite easily. The reason is that it allows applications to use the operating system to best advantage, in a standard way that permits faster and simpler development – something that is increasingly necessary in today's competitive commercial environment.

To achieve these aims, the runtime framework implements many of the features that the programmer, or the specific programming language environment, had to provide themselves. This includes things like automatic garbage collection, rich libraries of reusable objects to meet the needs of the most common tasks, and improved security for applications. This last point, of course, is becoming more important with the spread of networked applications – especially those that run over the Internet.

A Common Intermediate Language

However, one of the biggest advantages that the runtime framework provides is a language-neutral execution environment. All code, irrespective of the source language, is compiled automatically into a standard **intermediate language (IL)** – either on command or when first executed (in the case of ASP+). The runtime framework then creates the final binary code that makes up the application and executes it. The compiled IL code is used for each request until the source code is changed, at which point the cached version is invalidated and discarded.

So, whether you use Visual Basic, C#, JScript, Perl or any of the other supported languages, the intermediate code that is created is (or should be) identical. And the caching of the final binary object code improves efficiency and scalability at runtime as well.

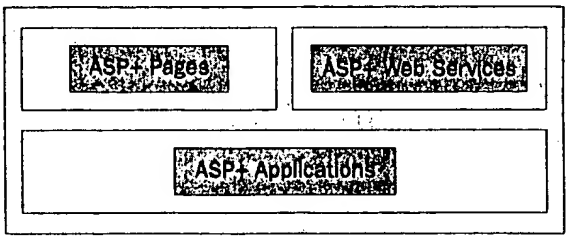
> *C# is the new language from Microsoft especially designed for use with the Next Generation Web Services framework and ASP+. It combines the power and efficiency of C++ with the simplicity of Visual Basic and JScript.*

One thing that this achieves is the ability to call from one language to another, and even inherit from objects created in one language and modify them within another language. For example, you can inherit an object that is written in C# in your VB program and then add methods or properties, or over-ride existing methods and properties. In fact, parts of the framework, and the entire ASP+ object model, are now implemented internally using C# rather than C++.

So, the new runtime framework introduces a true multi-language platform for programming any kind of application. As most of our current development is in the area of distributed applications, especially Internet- and Intranet-based applications, many the new features are directly aimed at this type of development.

The Web Application Infrastructure

The three sections shown highlighted in the previous diagram (and repeated in the next diagram) are those that implement ASP+ itself, and which we're interested in for this book:



Together, these three sections implement the **Web Application Infrastructure** – the topic that we are concerned with in this book. Together with the new runtime framework, it provides a range of exciting new features:

User Interface Support

As part of the ASP+ libraries, there is a host of intelligent server-based rich controls for building Web-based user interfaces quickly and simply. They can output HTML 3.2 code for down-level browsers, while taking advantage of the runtime libraries for enhanced interactivity on richer clients such as Internet Explorer 4 and above.

These server-based controls can be also be reused to build controls composed of other controls, inheriting implementation and logic from these constituent controls.

Data Access Support

The NGWS framework provides a new version of ADO, called **ADO+**, which offers integrated services for accessing data – regardless of the format or location of that data. ADO+ presents an object-oriented view of relational data, giving developers quick and easy access to data derived from distributed sources.

ADO+ also improves support for, and to some extent relies on, XML. ADO+ can automatically persist and restore recordsets (or **datasets** as they are now called) to and from XML. As we'll see, this is particularly useful when passing data around the Web using ASP+ Web Services.

Scalability for Distributed Applications

Two of the major requirements for any Web-based application are a robust operating platform, and scalability to allow large numbers of multiple concurrent requests to be handled. The NGWS runtime provides these features by allowing automatic error and overload detection to restart and manage the applications and components that are in use at any one time. This prevents errant code or memory leaks from soaking up resources and bringing the server to a halt.

There are also new and updated system and infrastructure services, including automatic memory management and garbage collection, automatic persistence and marshaling, and evidenced based security. Together these features provide for more scalable and reliable resource allocation and application processing.

Existing Software and Training Investments

Despite all the changes to the core operating system and runtimes, great care has been taken to maintain backward compatibility with earlier versions of Windows, COM and ASP. In most cases, existing applications, COM and COM+ components, ASP pages, and other scripts and executables work under the NGWS runtime. Alternatively, you can update them in your own time as your business requirements demand.

> **All ASP+ pages have the `.aspx` file extension, and this is mapped to the ASP+ runtime framework. This allows pages that have the `.asp` file extension to run unchanged under the existing ASP runtime.**

How is ASP+ Different from ASP?

Having seen in outline how ASP+ is now an integral part of the operating system, we need to look at the other aspect. How, and why, is ASP+ different to earlier version of ASP? And just **how** different is it? Well, if you just want to run existing pages and applications, you probably won't notice the differences much at all. However, once you open up the ASP+ SDK or Help files, you'll see a whole new vista of stuff that doesn't look the least bit familiar.

Don't panic! We'll work through the main differences next. We'll start with a look at why Microsoft has decided that we need a new version of ASP, and how it will help you, as a developer, to meet the needs of the future when creating Web sites and applications. We'll follow this with a checklist of the major new features of ASP+, then examine each one in a little more detail. The remainder of the book then covers the new features one by one, explaining how you can use them.

Why Do We Need a New Version of ASP?

In the Introduction to this book, we listed the main motivations that Microsoft had when designing and developing ASP+. After all, considering that ASP has been so successful, why do we need a new version? There are really four main issues to consider:

- Currently, ASP can only be scripted using the basic non-typed languages such as VBScript and JScript (unless you install a separate language interpreter). While ASP does parse and cache the code for the page the first time it is executed, the limitations prevent more strongly-typed languages like Visual Basic and C++ from being used where this would be an advantage. ASP+ provides a true language-neutral execution framework for Web applications to use.
- It is also very easy to create huge ASP pages containing a spaghetti-like mixture of code, HTML, text, object declarations, etc. And it's hard to re-use code, unless you place it in separate 'include' files – not the best solution. In many environments, developing a Web application utilizes the skills of a wide range of professionals, for example, you have programmers writing the code, and designers making the HMTL look good. Having both the code and the content intermixed in a single file that both of these groups need to operate on makes it difficult for them to work together. ASP+ allows true separation of code and content.
- In previous versions of ASP, you have to write code to do almost anything. Want to maintain state in form fields? Write code. Want to validate data entered on the client? Write code. Want to emit some simple data values? Write code. Want to cache regions of pages to optimize performance? Write code. ASP+ introduces a real component model, with server-based controls and an event driven execution paradigm similar in concept to the way that a Visual Basic 'Form' works now. The new ASP+ server controls are declarative (i.e. you only have to declare them in order to get them to do something), and so you actually write less code – in fact, in many situations you don't have to write any code at all!
- The world out there is changing. The proportion of users on the Web that will access your site through an 'Internet device' such as a mobile cellular phone, personal digital assistant (PDA), TV set-top box, games console, or whatever else, will soon be greater that the number using a PC and a traditional Web browser. This means that we probably have to be prepared to do more work at the server to tailor our pages to suit a specific device. We'll also have to create the output in a whole new range of formats such as the Wireless Markup Language (WML). And, in addition to creating WML for rendering, new Internet devices and business applications are going to want to be able to send and receive XML data from Web applications. Doing this today from ASP requires you to manually use an XML parser, convert data to and from XML schemas, etc. ASP+ Web Services makes it much easier.

Besides all of this, the rapidly changing nature of distributed applications requires faster development, more componentization and re-usability, easier deployment, and wider platform support. New standards such as the Simple Object Access Protocol (SOAP), and new commercial requirements such as business-to-business (B2B) data interchange, require new techniques to be used to generate output and communicate with other systems. Web applications and Web sites also need to provide a more robust and scalable service, which ASP+ provides through proactive monitoring and automatic restarting of applications when failures, memory leaks, etc. are discovered.

So, to attempt to meet all these needs, ASP has been totally revamped from the ground up into a whole new programming environment. While there are few tools available to work with it just yet, Visual Studio 7.0 will be providing full support to make building ASP+ applications easy (both ASP+ Pages and ASP+ Services).

The rich, component based, event driven programming model is specifically designed to be 'tools friendly', and this support will be available for all Visual Studio languages – including VB, C++, and C#. And you can be sure that third party tool manufacturers will not be far behind.

The Big Advantages with ASP+

The biggest challenges facing the Web developer today must be the continued issues of browser compatibility, and the increasing complexity of the pages that they have to create. Trying to build more interactive pages that use the latest features of each browser, whilst still making sure that the pages will work on all the popular browsers, is a nightmare that refuses to go away.

And, of course, it will only get worse with the new types of Internet device that are on the way, or here already. In particular, trying to build pages to offer the same user-level capability to cellular phones as to traditional browser clients is just about impossible. The text-only 12-character by 3-line display of many cellular phones does tend to limit creativity and user interaction.

One obvious solution is to create output that is targeted at each specific client dynamically – or create multiple versions of the same site, one for each type of client. The second option is not attractive, and most developers would prefer the first one. However, this implies that every hit from every user will require some server-side processing to figure out what output to create.

If this is the case, why not automate much of the process? To this end, ASP+ introduces the concept of server controls that encapsulate common tasks and provide a clean programming model. They also help to manage the targeting of all the different types of client.

Server-side HTML Controls – Less Code to Write

ASP has always provided the opportunity to execute components on the server, and these components can generate sections of the page that is returned to the user. ASP+ extends this concept through **server controls**. All that's required to turn any HTML element into a server control is the addition of an extra attribute: `runat="server"`.

Any HTML element in a page can be marked this way, and ASP+ will then process the element on the server and can generate output that suits this specific client. And, as a by-product, we can do extra tricks – in particular with HTML `<form>` and the associated form control elements, where we can create the code to manage state during round trips to the server. This makes the programming experience less monotonous and dramatically more productive.

While the concept of having HTML elements that execute on the server may at first seem a little strange, as you'll see it adds a whole new layer of functionality to the pages, and makes them easier to write at the same time. What more could a programmer want?

The Problems with Maintaining State

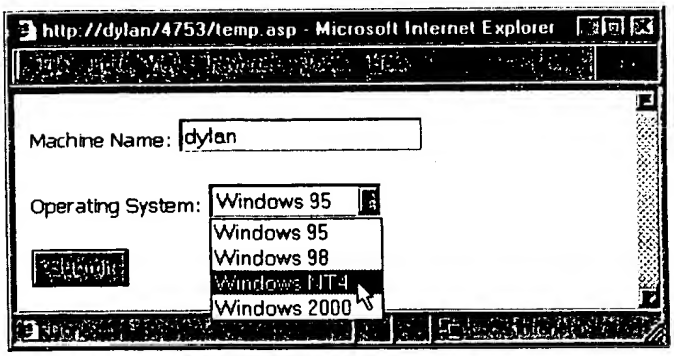
One of the most cumbersome tasks when creating interactive Web sites and applications is managing the values passed to the server from HTML form controls, and maintaining the values in these controls between page requests. So one of the core aims of ASP+ is to simplify this programming task. This involves no extra effort on the part of the programmer, and works fine on all browsers that support basic HTML and above.

Take a look at the following section of code. This creates a simple form using HTML controls where the user can enter the name of a computer and select an operating system. OK, so this isn't a terribly exciting example in itself, but it illustrates a pretty common scenario used by almost every web application out there today. When the form page is submitted to the server, the values the user selected are extracted from the `Request.Form` collection and displayed with the `Response.Write` method. The important parts of the page are highlighted in the code listing:

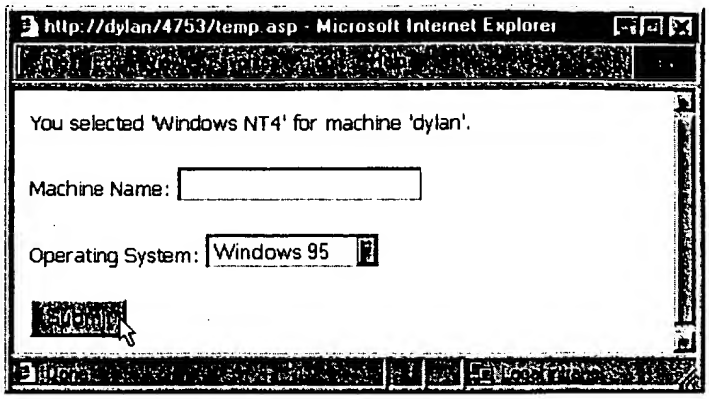
```
<html>
  <body>
    <%
    If Len(Request.Form("selOpSys")) > 0 Then
      strOpSys = Request.Form("selOpSys")
      strName = Request.Form("txtName")
      Response.Write "You selected '" & strOpSys _
                  & "' for machine '" & strName & "'."
    End If
    %>
    <form action="pageone.asp" method="post">
      Machine Name:
      <input type="text" name="txtName">
      <p />
      Operating System:
      <select name="selOpSys" size="1">
        <option>Windows 95</option>
        <option>Windows 98</option>
        <option>Windows NT4</option>
        <option>Windows 2000</option>
      </select>
      <p />
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Although this is an ASP page (the file extension is `.asp` rather than `.aspx`), it will work just the same under ASP+ if we changed the extension to `.aspx`. Remember that the two systems can quite freely co-exist on the same machine, and the file extension just determines whether ASP or ASP+ processes it.

This screenshot shows what it looks like in Internet Explorer 5. When the user clicks the Submit button to send the values to the server, the page is reloaded showing the selected values. Of course, in a real application, some the values would probably be stored in a database, or be used to perform some application-specific processing – for this example we're just displaying them in the page:



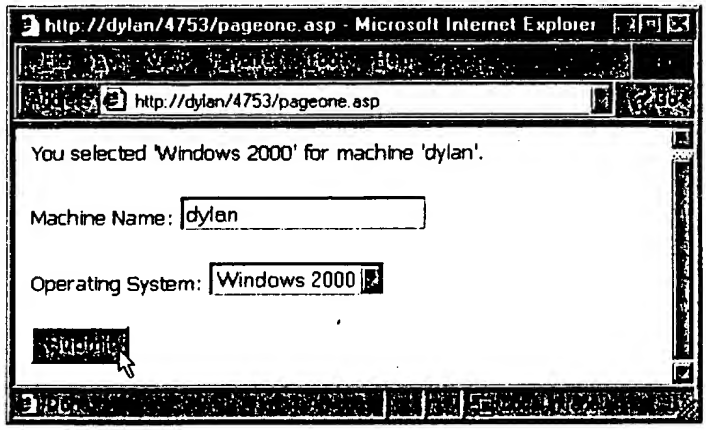
One problem is that the page does not maintain its **state**, in other words the controls return to their default values. The user has to re-enter them to use the form again. You can see this is the next screenshot:



To get round this situation, we have to add extra ASP code to the page to insert the values into the controls when the page is reloaded. For the text box, this is just a matter of setting the `value` attribute with some inline ASP code, using the `HTMLEncode` method to ensure that any non-legal HTML characters are properly encoded. However, for the `<select>` list, we have to do some work to figure out which value was selected, and add the `selected` attribute to that particular `<option>` element. The changes required are highlighted below:

```
<html>
  <body>
    <%
    If Len(Request.Form("selOpSys")) > 0 Then
      strOpSys = Request.Form("selOpSys")
      strName = Request.Form("txtName")
      Response.Write "You selected '" & strOpSys _
                  & "' for machine '" & strName & "'."
    End If
    %>
    <form action="pageone.asp" method="post">
      Machine Name:
      <input type="text" name="txtName"
             value="<% = Server.HTMLEncode(Request("txtName")) %>">
      <p />
      Operating System:
      <select name="selOpSys" size="1">
        <option
        <% If strOpSys = "Windows 95" Then Response.Write " selected" %>
        >Windows 95</option>
        <option
        <% If strOpSys = "Windows 98" Then Response.Write " selected" %>
        >Windows 98</option>
        <option
        <% If strOpSys = "Windows NT4" Then Response.Write " selected" %>
        >Windows NT4</option>
        <option
        <% If strOpSys = "Windows 2000" Then Response.Write " selected" %>
        >Windows 2000</option>
      </select>
      <p />
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

Now, when the page is reloaded, the controls maintain their state and show the values the user selected:



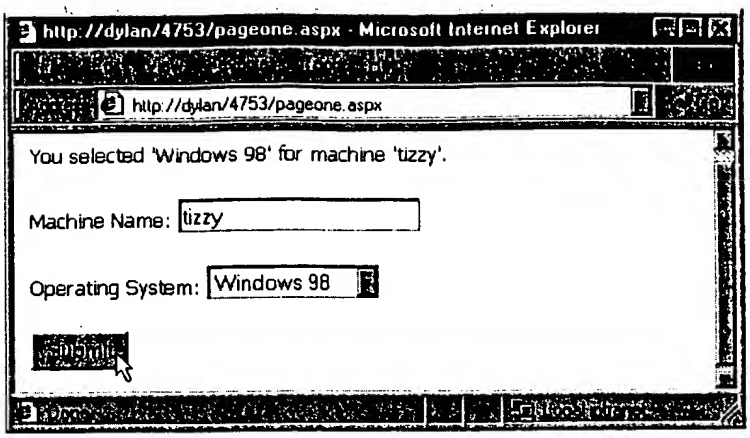
This page, named `pageone.asp`*, is in the* `Chapter01` *directory of the samples available for the book. You can download all the sample files from our Web site at http://www.wrox.com.*

Controls that Automatically Maintain Their State

So, how does ASP+ help us in this commonly met situation? The next listing shows the changes required for taking advantage of ASP+ server controls that automatically preserve their state. We still use the `Response.Write` method to display the selected values. However, this time some of the elements on the page have the special `runat="server"` attribute added to them. When ASP+ sees these elements, it processes them on the server and creates the appropriate HTML output for the client:

```
<html>
  <body>
    <%
    If Len(Request.Form("selOpSys")) > 0 Then
      strOpSys = Request.Form("selOpSys")
      strName = Request.Form("txtName")
      Response.Write("You selected '" & strOpSys _
                     & "' for machine '" & strName & "'.")
    End If
    %>
   <form runat="server">
      Machine Name:
      <input type="text" id="txtName" runat="server">
      <p />
      Operating System:
      <select id="selOpSys" size="1" runat="server">
        <option>Windows 95</option>
        <option>Windows 98</option>
        <option>Windows NT4</option>
        <option>Windows 2000</option>
      </select>
      <p />
      <input type="submit" value="Submit">
    </form>
  </body>
</html>
```

You can clearly see how much simpler this ASP+ page is than the last example. When loaded into Internet Explorer 5 and the values submitted to the server, the result appears to be just the same:



This page, named `pageone.aspx`*, is in the* `Chapter01` *directory of the samples available for the book. You can download all the sample files from our Web site at http://www.wrox.com.*

How Do the Server-Side Controls Work?

How is this achieved? The key is the `runat="server"` attribute. To get an idea of what's going on, take a look at the source of the page from within the browser. It looks like this:

```
<html>
  <body>
    You selected 'Windows 98' for machine 'tizzy'.
    <FORM name="ctrl0" method="post" action="pageone.aspx" id="ctrl0">
    <INPUT type="hidden" name="__VIEWSTATE" value="a0z1741688109__x">
      Machine Name:
      <INPUT type="text" id="txtName" name="txtName" value="tizzy">
      <p />
      Operating System:
      <SELECT id="selOpSys" size="1" name="selOpSys">
        <OPTION value="Windows 95">Windows 95</OPTION>
        <OPTION selected value="Windows 98">Windows 98</OPTION>
        <OPTION value="Windows NT4">Windows NT4</OPTION>
        <OPTION value="Windows 2000">Windows 2000</OPTION>
      </SELECT>
      <p />
      <input type="submit" value="Submit">
    </FORM>
  </body>
</html>
```

We wrote this ASP+ code to create the `<form>` in the page:

```
<form runat="server">
   ...
</form>
```

When the page is executed by ASP+, the output to the browser is:

```
<FORM name="ctrl0" method="post" action="pageone.aspx" id="ctrl0">
   ...
</FORM>
```

You can see that the `action` and `method` attributes are automatically created by ASP+ so that the values of the controls in the form will be `POST`ed back to the same page. ASP+ also adds a unique `id` and `name` attribute to the form as we didn't provide one. However, if you do specify these, the values you specify will be used instead.

> **If you include the `method="GET"` attribute, the form contents are sent to the server as part of the query string instead, as in previous versions of ASP, and the automatic state management will no longer work.**

Inside the form, we wrote this ASP+ code to create the text box:

```
<input type="text" id="txtName" runat="server">
```

The result in the browser is this:

```
<INPUT type="text" id="txtName" name="txtName" value="tizzy">
```

You can see that ASP+ has automatically added the `value` attribute with the text value that was in the control when the form was submitted. It has also preserved the `name` attribute we provided, and added an `id` attribute with the same value.

For the `<select>` list, we wrote this code:

```
<select id="selOpSys" size="1" runat="server">
  <option>Windows 95</option>
  <option>Windows 98</option>
  <option>Windows NT4</option>
  <option>Windows 2000</option>
</select>
```

ASP+ obliged by outputting this HTML, which has a `selected` attribute in the appropriate `<option>` element:

```
<SELECT name="selOpSys" id="selOpSys" size="1">
  <OPTION value="Windows 95">Windows 95</OPTION>
  <OPTION selected value="Windows 98">Windows 98</OPTION>
  <OPTION value="Windows NT4">Windows NT4</OPTION>
  <OPTION value="Windows 2000">Windows 2000</OPTION>
</SELECT>
```

Again, a unique `id` attribute has been created, and the `<option>` elements have matching `value` attributes added automatically. (If we had provided our own value attributes in the page, however, these would have been preserved.)

The Page VIEWSTATE

The other change is that ASP+ has automatically added a `HIDDEN`-type control to the form:

```
<INPUT type="hidden" name="__VIEWSTATE" value="a0z1741688109__x">
```

This is how ASP+ can store ambient state changes of a page across multiple requests – i.e. things that don't automatically get sent back and forth between the browser and server between Web requests. For example, if the background color of a server control had been modified it would use the VIEWSTATE hidden field to remember this between requests. The VIEWSTATE field is used whenever you post back to the originating page. In Chapter 2, we discuss this topic in more detail.

> **So, as you can see, there really aren't any 'magic tricks' being played. It's all standard HTML, with *no* client-side script libraries, and *no* ActiveX controls or Java applets. An equally important point is that *absolutely no state* is being stored on the server. Instead, values are simply posted to the server using standard methods. Values are preserved and maintained across requests simply by the server controls modifying the HTML before the pages are sent to the client.**

Server-side Code In ASP+

To display the values in the page, we used code that is very similar to that we used in the ASP example earlier on:

```
...
If Len(Request.Form("selOpSys")) > 0 Then
  strOpSys = Request.Form("selOpSys")
  strName = Request.Form("txtName")
  Response.Write("You selected '" & strOpSys _
                 & "' for machine '" & strName & "'.")
End If
...
```

However, one of the other great features of ASP+ and server controls is that they are available to the code running on the server that is creating the page output. The ASP+ interpreter insists that each one has a unique id attribute, and therefore *all* the server controls (i.e. the elements that have the runat="server" attribute) will be available to code against. This means that we no longer have to access the Request collection to get the values that were posted back to the server from our form controls – we can instead refer to them directly using their unique id:

```
...
If Len(selOpSys.value) > 0 Then
  Response.Write("You selected '" & selOpSys.value _
                 & "' for machine '" & txtName.value & "'.")
End If
...
```

Visual Basic Code In ASP+

In the ASP page we've just seen, the script was assumed to be VBScript (we didn't specify this, and VBScript is the default unless you change the server settings). In ASP+, there is no support for VBScript. Instead, the default language is Visual Basic ("VB"), which is a superset of VBScript. So, our code is being compiled into IL and executed by the runtime.

The compiler and runtime for Visual Basic that is included with ASP+ is the new version 7.0 (good news – you don't need to buy a separate copy!). There are a few implications in this, which we summarize in Appendix **B** of this book.

The most important thing to note straight away is that all method calls in VB7 must have the parameter list enclosed in parentheses (much like JScript and JavaScript). In VBScript and earlier versions of VB, this was not required – and in some cases produced an error. You can see that we've enclosed the parameter to the `Response.Write` method in parentheses in our example.

Secondly, VB7 has no concept of 'default' methods or 'default' properties, so we now must provide the method or property name. You'll probably come across this first when working with ADO recordsets where the syntax must be:

```
fieldvalue = objRecordset.Fields("fieldname").value
```

Server-side Event Processing – A Better Code Structure

Of course, if we are going to have HTML elements that execute on the server, why not extend the concept even more? ASP+ changes each page into a server-side object, and exposes more properties, methods and events that can be used within your code to create the content dynamically. Each page becomes a tree of COM+ objects that can be accessed and programmed individually as required.

Using Server-side Control Events

To see how we can take advantage of this to structure our pages more elegantly, take a look at the following code. It shows the ASP+ page we used in our previous example, but with a couple of changes. This version of the page is named `pagetwo.aspx`:

```
<html>
  <body>
    <script language="VB" runat="server">
      Sub ShowValues(Sender As Object, Args As EventArgs)
        divResult.innerText = "You selected '" _
          & selOpSys.value & "' for machine '" _
          & txtName.value & "'."
      End Sub
    </script>
    <div id="divResult" runat="server"></div>
    <form runat="server">
      Machine Name:
      <input type="text" id="txtName" runat="server">
      <p />
      Operating System:
      <select id="selOpSys" size="1" runat="server">
        <option>Windows 95</option>
        <option>Windows 98</option>
        <option>Windows NT4</option>
        <option>Windows 2000</option>
      </select>
      <p />
      <input type="submit" value="Submit"
             runat="server" onserverclick="ShowValues">
    </form>
  </body>
</html>
```

Firstly, notice that we've replaced the inline ASP+ code with a `<script>` section that specifies VB as the language, and includes the `runat="server"` attribute. Inside it, we've written a Visual Basic function named `ShowValues`. In ASP+, functions and subroutines must be placed inside a server-side `<script>` element, and not in the usual `<%...%>` script delimiters – we'll look at this in more detail in the next chapter.

We've also added an HTML `<div>` element to the page, including the `runat="server"` attribute. So this element will be created on the server, and is therefore available to code running there. When the VB subroutine is executed, it sets the `innerText` property of this `<div>` element.

Notice also how it gets at the values required (e.g. those submitted by the user). Because the text box and `<select>` list also run on the server, our code can extract the values directly by accessing the `value` properties of these controls. When the page is executed and rendered on the client, the `<div>` element that is created looks like this:

```
<div id="divResult">You selected 'Windows NT4' for machine 'lewis'.</div>
```

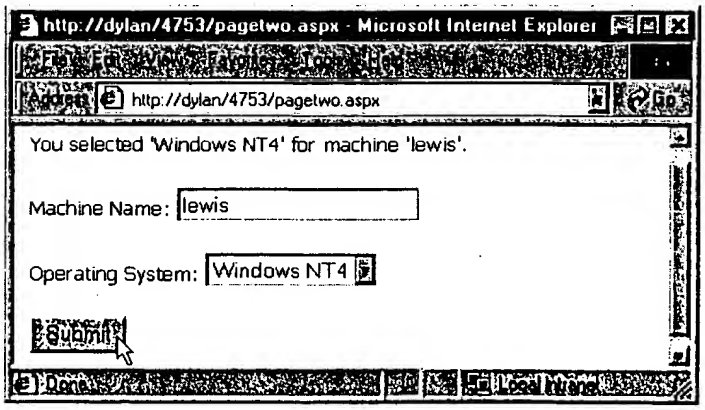
Connecting Server-side Control Events To Your Code

By now you should be asking how the VB subroutine actually gets executed. Easy – in the `<input>` element that creates the Submit button, we added two new attributes:

```
<input type="submit" value="Submit"
       runat="server" onserverclick="ShowValues">
```

The `runat="server"` attribute converts the HTML element into a server-side control that is 'visible' and therefore programmable within ASP+ on the server. The `onserverclick="ShowValues"` attribute then tells the runtime that it should execute the `ShowValues` subroutine when the button is clicked. Notice that the server-side event names for HTML controls include the word `"server"` to differentiate them from the client-side equivalents (i.e. `onclick`, which causes a client-side event hander to be invoked).

The result is a page that works just the same as the previous example, but the ASP+ source now has a much more structured and 'clean' format. It makes the code more readable, and still provides the same result – without any client-side script or other special support from the browser. If you view the source code in the browser, you'll see that it's just the same:



This page, named `pagetwo.asp`, is in the `Chapter01` directory of the samples available for the book. You can download all the sample files from our Web site at http://www.wrox.com.

You'll see how we can improve the structure even more, by separating out the code altogether, in Chapter 2. And, even better (as with earlier versions of ASP) we can add our own custom components to a page or take advantage of a range of server-side components that are provided with the NGWS framework.

Many of these can be tailored to create output specific to the client type, and controlled by the contents of a template within the page.

The ASP+ Application Framework

As a by-product of the modularization of ASP+, developers can also access the underlying runtime framework if they need to work at a lower level than the ASP+ page itself. As well as the information made available through the traditional ASP objects such as `Form`, `QueryString`, `Cookies` and `ServerVariables`, developers can also access the underlying objects that perform the runtime processing.

These objects include the entire page context, the **HTTP Modules** that process the requests, and the **Request HTTP Handler** objects. It is also possible to access the raw data streams, which are useful for managing file uploads and other similar specific tasks. We look at this whole topic in detail in Chapter 6.

Compilation and Rich Language Support – Enhanced Performance

When a page or Web service is first activated by the client, ASP+ dynamically compiles the code, caches it, and then reuses this cached code for all subsequent requests until the original page is changed – at which point the compiled code version is invalidated and removed from the cache. You can see this as a delay the first time that an ASP+ page is executed, while the response to subsequent requests is almost instant.

Because the compilation is to the **intermediate language** (rather than the processor-level binary code), any language can be used as long as the compiler outputs code in this intermediate language. In fact, a number of independent vendors are already working on different languages (including Cobol).

And, because the intermediate language code is common across languages, each language can inherit from all others and call routines that were originally written in all the other languages. The efficient component management services provided by the runtime also ensure that the compiled code in the page executes much more efficiently than would be possible using the earlier versions of ASP.

A Checklist of the New Features

The major features that ASP+ provides over earlier versions of ASP are:

- **Pages** that use the new server-side controls to automate state management in the page, and reduce the code you have to write. Because ASP+ pages have a programming model similar to VB Forms, we also refer to ASP+ pages as Web Forms.
- **HTML Server-side Controls** can be used to generate the HTML elements in the page output, and allow code to be used to set the properties (i.e. attributes) of these controls at runtime. They also allow events raised by the elements to be detected and appropriate code executed on the server in response to these events.
- **Rich Controls** that run on the server can be used to create more complex HTML elements and objects in the page output. ASP+ includes a calendar control and a range of grid, table, and list controls. These controls can also take advantage of server-side data binding to populate them with values.

- **Web Services** allow developers to create classes that are generally not rendered as visible output, but instead provide services to the client. For example, they can include functions that return specific values in response to a request.
- **Configuration and Deployment** is now easier, with the use of human-readable XML-format configuration files. Components no longer need to be registered on the server (no more `regsvr32`!), and applications can be deployed using file copy commands, the FrontPage server extensions or FTP.
- **Application and Session State Management** is extended to provide a persistent and more scalable environment for storing values relevant to specific clients and applications. In particular, a user's session state can now easily be maintained in a Web farm.
- **Error Handling, Debugging, and Tracing** features have been greatly extended and improved. Each page can have its own 'error page', and can also display values that are used in the page code as it executes – providing a 'trace' facility. Debugging can also be carried out across languages – allowing you to single step seamlessly from one language to another, for example from code in a VB page into a C++ component.
- **Security Management Features** now allow many different kinds of login and user authentication to be used. Instead of the default browser login prompt (with Windows 2000 and NTLM authentication), custom login pages can be created and managed by ASP+. It is also easier to manage users depending on the role or group they belong to.
- **Custom Server-side Caching** allows developers to store all kinds of values and objects locally on the server for use in ASP+ pages. The runtime can also cache the *output* from ASP+ pages. This can provide a huge performance boost in situations where a dynamically created page is the same for many visitors, such as a product catalog.
- **A Range of Useful Components** are shipped with ASP+. These class libraries can help to make writing Web applications easier. Examples include: the 'SendMail' component, encryption/decryption components, components for defining custom performance counters, components for reading and writing to the NT event log, components for working with MSMQ, network access components (replacements for WinInet), data access components, etc.

Features such as the intrinsic `Request` and `Response` objects (and the `Form`, `QueryString`, `Cookies`, and `ServerVariables` collections that they implement) are compatible with earlier versions of ASP. However, they have gained a lot of new properties and methods that make it easier to build applications. There is also access to the `ObjectContext` object for use by any existing ASP components. However, there are some new methods and properties available for the intrinsic ASP objects, as well as other issues that affect how existing pages, components and applications perform under ASP+. See Appendix A for more details.

ASP+ Pages

ASP+ Pages are described in detail in Chapters **2**, **3** and **4**. The four big advantages that ASP+ Pages provide are:

- **Controls can encapsulate reusable functionality**. This allows them to automate and simplify a lot of the common programming tasks, such as state management, validation, data manipulation, etc. that require specific coding in previous versions of ASP.
- **Code is 'cleaner' and easier to read**. The encapsulation of code in server controls, combined with the ability to use proper event handling techniques in your pages, allows a more structured design. Reusability of previously tested and optimized controls also means that development is faster.

- There is **better support for code and user interface development tools.** These can provide true WYSIWYG editing of pages, management of controls properties, and easy access to methods and events of the controls.
- It **removes the dependency of ASP on non-typed script languages** such as VBScript and JScript. Code can be written in any of the ASP+ supported languages, such as Visual Basic, C++, C#, Perl, etc. It can also be written as separate modules and used within the page, rather than as traditional inline code.

The ASP+ Control Families

ASP+ provides a series of new server controls that can be instantiated within an ASP+ page. From the developer's point of view, the advantage of using these controls is that server-side processing can be carried out on events raised by client-side controls.

The server controls provided with ASP+ fall into four broad categories or 'families':

- **Intrinsic controls** that create HTML-style elements on the client. They can create intelligent controls that automatically maintain state and provide extra features, or just plain HTML elements.
- **List controls** that can automatically produce lists of all kinds on the client. In conjunction with server-side data binding, they can also populate the lists with data from databases, XML files, etc. – using only a few lines of code.
- **Rich controls** output client-side HTML, and in some cases client-side script as well, in order to create more complex types of controls or interface elements on the client. An example is the 'Calendar' control, which detects the browser type and creates corresponding code to complement the features of that browser.
- **Validation controls** are non-visible controls that make it easy to do client-side or server-side validation when creating forms for the user to fill in and post back to the server. There is a range of these controls, allowing complex validation to be carried out easily.

All of these controls are designed to produce output that can run on **any** Web browser (you'll see this demonstrated in several places within the book). There are no client-side ActiveX controls or Java applets required. We'll look at each of these control types in more detail next.

The ASP+ Intrinsic Controls

In the example we looked at earlier in this chapter, we saw how ASP+ provides a series of **Intrinsic Controls** that are intelligent. In other words, they can be executed on the server to create output that includes event handling and the maintenance of **state** (the values the controls display). In Chapters 2, 3, and 4, we look at how we can use these controls in more detail, and explore their various capabilities.

However, to overview the aims of the new ASP+ Intrinsic Controls, we can say that they serve three main purposes:

- They allow the developer to interact with the control on the server when the page is being created, in particular by setting the values of properties or reacting to events that are raised on the client.
- They automatically create the appropriate HTML to preserve their current state, so that they display the correct values as selected by the user when the page is reloaded – without requiring the developer to write code to do this.

- They make development simpler and faster, and promote reusability and better page design and structure by encapsulating the repetitive code required for these tasks within the control.

The basic intrinsic controls are used by simply inserting the equivalent HTML into the page, just as you would in earlier versions of ASP, but adding the `runat="server"` attribute. The elements that are implemented as specific objects in the preview version of ASP+ are:

`<table>`	`<tr>`	`<th>`	`<td>`
`<form>`	`<input>`	`<select>`	`<textarea>`
`<button>`	`<a>`	`<img>`	

As in HTML, the `<input>` server control depends on the value of the `type` attribute. The output that the control creates is, of course, different for each value.

All other HTML elements in an ASP+ page that are marked with the `runat="server"` attribute are handled by a single generic HTML server control. It creates output based simply on the element itself and any attributes you provide or set server-side when the page is being created.

There is also a set of new ASP+ controls that can be defined within the page, and which are prefixed with the namespace 'asp'. These controls expose properties that correspond to the standard attributes that are available for the equivalent HTML element. As with all server controls, you can set these properties during the server-side `Load` events of the page, or add them as attributes in the usual way, but using the special property names. When rendered to the client, the properties are converted into the equivalent HTML syntax.

For example, to create an instance of a `ListBox` control, we can use:

```
<asp:ListBox visibleItems="3" runat="server">
  <asp:ListItem>Windows 98</asp:ListItem>
  <asp:ListItem>Windows NT4</asp:ListItem>
  <asp:ListItem>Windows 2000</asp:ListItem>
</asp:ListBox>
```

At runtime (in the preview version) the ASP+ code above creates the following HTML, and sends it to the client:

```
<SELECT name="ListBox0" size="3">
    <OPTION value="Windows 98">Windows 98</OPTION>
    <OPTION value="Windows NT4">Windows NT4</OPTION>
    <OPTION value="Windows 2000">Windows 2000</OPTION>
</SELECT>
```

A summary of the 'asp'-prefixed intrinsic controls looks like this:

ASP+ Intrinsic Control	HTML Output Element
`<asp:Button>`	`<input type="submit">`
`<asp:LinkButton>`	`<a href="jscript:__doPostBack(...)">...<a>`
`<asp:ImageButton>`	`<input type="image">`

Table continued on following page

ASP+ Intrinsic Control	HTML Output Element
<asp:HyperLink>	...
<asp:TextBox>	<input type="text" value="...">
<asp:CheckBox>	<input type="checkbox">
<asp:RadioButton>	<input type="radio">
<asp:DropDownList>	<select>...</select>
<asp:ListBox>	<select size="...">...</select>
<asp:Image>	<img src="...">
<asp:Label>	...
<asp:Panel>	...
<asp:Table>	<table>...</table>
<asp:TableRow>	<tr>...</tr>
<asp:TableCell>	<td>...</td>

These controls provide more standardized property sets than the HTML controls, and make it easier to implement tools that can be used for designing and building ASP+ pages and applications.

The ASP+ List Controls

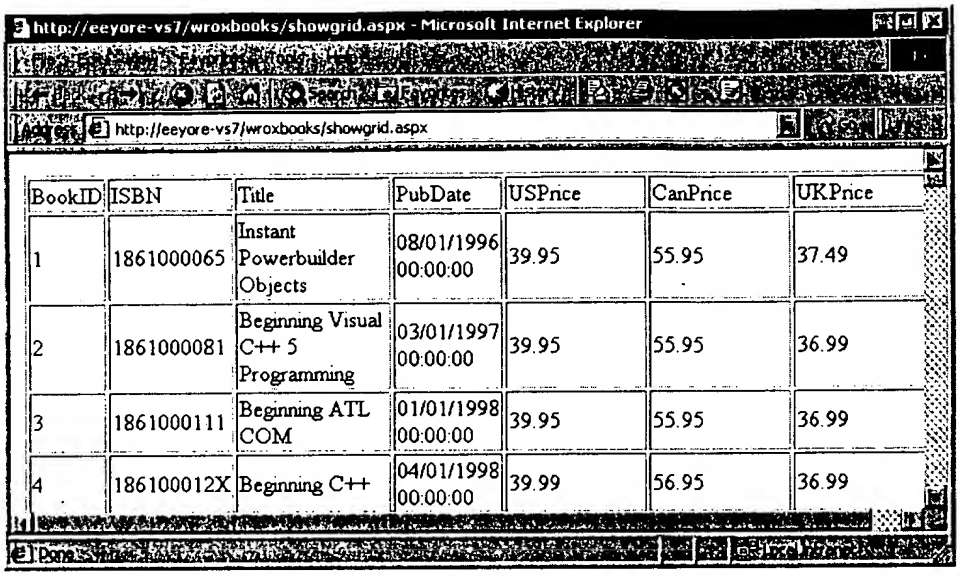
Many day-to-day tasks involve the listing of data in a Web page. In general, this data will be drawn from a data store of some type, perhaps a relational database. The aim of the ASP+ **List Controls** is to make building these kinds of pages easier. This involves encapsulating the functionality required within a single control or set of controls, saving development time and making the task easier.

The server-side ASP+ controls that generate the user interface can also manage tasks such as paging the list, sorting the contents, filtering the list, and selecting individual items. Finally, they can use the new server-side data binding features in ASP+ to automatically populate the lists with data. The three standard controls that are used to create lists are the **Repeater**, **DataList**, and **DataGrid** controls.

The **Repeater** control is the simplest, and is used simply to render the output a repeated number of times. The developer defines templates that are used to apply styling information to the `header`, `item`, `footer` and `separator` parts of the output that is created by the control. To create a table, for example, the `header` and `footer` information is used (as you would expect) for the header and footer of the table (the `<thead>` and `<tfoot>` parts in HTML terms). The `item` template content is applied to every row of the table that is generated from the repeated values – probably the records from a data store of some type. There is also the facility to use an `alternatingItem` template to apply different styles to alternate rows. The `separator` information defines the HTML output that will be generated after each row and before the next one.

The **DataList** control differs from the **Repeater** control in that it provides some intrinsic formatting of the repeated data as well as accepting the same kinds of template values as the **Repeater** control. The **DataList** control renders additional HTML (outside of that defined in its templates) to better control the layout and format of the rendered list, providing features such as vertical/horizontal flow and style support. However, unlike the **Repeater** control, it can also be used to edit the values in the elements on demand, and detect changes made by the user.

The richest of the list controls is the **DataGrid** control. The output is an HTML table, and the developer can define templates that are used to apply styling information to the various parts of the table. As well as being able to edit the values in the table, the user can also sort and filter the contents, and the appearance is similar to that of using a spreadsheet. However, as the output from the control is simply HTML, it will work in all the popular browsers.



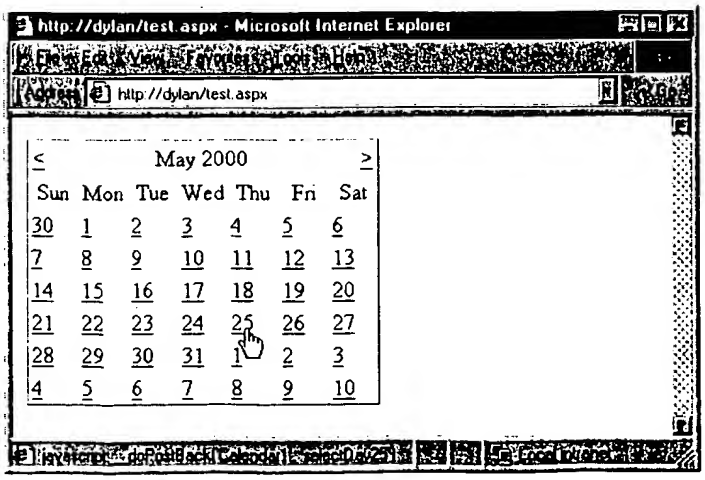
BookID	ISBN	Title	PubDate	USPrice	CanPrice	UKPrice
1	1861000065	Instant Powerbuilder Objects	08/01/1996 00:00:00	39.95	55.95	37.49
2	1861000081	Beginning Visual C++ 5 Programming	03/01/1997 00:00:00	39.95	55.95	36.99
3	1861000111	Beginning ATL COM	01/01/1998 00:00:00	39.95	55.95	36.99
4	186100012X	Beginning C++	04/01/1998 00:00:00	39.99	56.95	36.99

There are two other more specialized types of list control included with ASP+. These are the **RadioButtonList** and **CheckboxList** controls. In effect, they simply render on the client a list of HTML radio button or checkbox elements, with captions applied using span elements. However, you can easily specify the layout, i.e. if the items should appear listed horizontally across the page or vertically down it, or within an HTML table. You can also control the alignment of the text labels, and arrange for the list to automatically post back the selected values.

The ASP+ Rich Controls

The preview version of ASP+ ships with three **Rich Controls** that provide specific functions not usually available in plain HTML. Examples of these are the **Calendar** and **AdRotator** controls. Also due in later releases are **TreeView**, **ImageGenerator**, and other controls.

To give you some idea of what these custom controls can do, take a look at the screenshot below:



This was generated using this simple code:

```
<form runat="server">
  <asp:Calendar runat="server" />
</form>
```

The ASP+ Validation Controls

One common task when working with HTML forms in a Web application is the validation of values that the user enters. They may have to fall within a prescribed range, be non-empty, or even have the values from several controls cross-referenced to check that the inputs are valid.

Traditionally, this has been done using either client-side or server-side scripts, often specially written for each form page. In ASP+, a range of **Validation Controls** is included. These make it easy to perform validation checks – both client-side and server-side.

Three types of validation control are provided. The **RequiredFieldValidator** control, **CompareValidator** control, **RangeValidator** control, and **RegularExpressionValidator** control perform various types of checks to ensure that that a control contains a specific value, or matches the value in another control. The **CustomValidator** control passes the value that a user enters into a control to a specified client-side or server-side function for custom validation. The **ValidationSummary** control collects all the validation errors and places them in a list within the page.

Each one of these controls (except the **ValidationSummary** control) is linked to one or more HTML controls through the attributes you set for the validation control. They can automatically output the text or character string you specify when the validation fails. You can also use the `IsValid` method of the `Page` object to see if any of the validation controls detected an error, and provide custom error messages.

Some of the controls also detect the browser type, and can output code that performs validation client-side without requiring a round-trip to the server. If this is not possible, code to do the validation during the submission of the values to the server is output instead. We look in detail at how we can use these controls in Chapter 4.

You'll see most of the controls we've discussed described in more detail in Chapter 2. The more complex topics associated with them, such as templates, server-side data binding and ADO+ are covered in Chapter 3. Chapter 4 looks at the validation controls, and other advanced techniques in ASP+ pages. We also look at how you can build your own custom server-side controls in Chapter 7.

ASP+ Web Services

As the march of XML through traditional computing territory continues unabated, more and more of the ways that we are used to doing things are changing. One area is the provision of programmatic services that can be consumed by remote clients, especially where the server and client are running on different operating system platforms. The **Simple Object Access Protocol** (SOAP) is an XML grammar that allows clients to take advantage of the services provided by a remote server or application, by providing the requests in a standard format.

ASP+ includes support for the creation of suitable server or application objects that accept SOAP requests, and return the results in SOAP format. The technology is called **Web Services**, and allows developers to create these objects quickly and easily within the .NET framework.

These objects are also automatically discoverable and described to clients using the XML-based **Service Description Language** (SDL).

You simply create the object using normal server-side code as a `public class`, in any of the supported languages, and include one or more methods that the client can access marked with the special `[ WebMethod ]` indicator. These are known as custom attributes. Other methods that are not marked as such are private, and cannot be accessed by clients. No knowledge of COM+ or HTTP itself is required, and the source for these service objects is just text files.

Web Services allows custom business service objects to be created quickly and easily by ASP+ developers. The client can access them synchronously or asynchronously, using the `HTTP-GET`, `HTTP-POST` or `HTTP-SOAP` methods that provide extra flexibility. As with other objects in ASP+, the source is compiled, cached and executed under the runtime. We explore the whole concept of Web Services in Chapter 5.

ASP+ Configuration and Deployment

In earlier versions of ASP, a file named `global.asa` could exist in the root directory of a virtual application. This file defined global variables and event handlers for the application. However, all other configuration details for the entire Web site were made in the Internet Services Manager snap-in to the MMC. The settings made in Internet Services Manager are stored in the IIS metabase, which is a server-based machine-readable file that specifies the whole structure of the Web services for that machine.

This has at least one major disadvantage. It requires the administrator or developer to access the metabase using either Internet Services Manager (it can be used remotely to access another machine on the LAN), the equivalent HTML pages, or custom pages that access the metabase through the Active Directory Services Interface (ADSI).

The Global Configuration File – config.web

In ASP+, all the configuration details for all Web applications are kept in human-readable files named `config.web`. The default `config.web` file is in the `Program Files\COM20SDK\` directory and this specifies the settings that apply to any applications or directories that do not over-ride the defaults. The standard format for the configuration files is XML, and each application inherits the settings in the default `config.web` file.

The `config.web` file specifies a whole range of settings for the application, including the HTTP Modules and Request Handlers that are to be used to handle each request. This provides a completely extensible and flexible architecture, allowing non-standard HTTP protocol handling to be carried out if required. We examine configuration files and their use in Chapter 6.

The Application Definition File – global.asax

As in ASP 2.0 and 3.0, it is also possible to use a definition file that specifies the actions to take when an application starts and ends, and when individual user sessions start and end. This file is named `global.asax` (note the `.asax` file extension), and is stored in the root directory for each application.

The existing ASP event handlers `Application_OnStart`, `Application_OnEnd`, `Session_OnStart`, and `Session_OnEnd` are supported in `global.asax`, as well as several new events such as `Application_BeginRequest`, `Security_OnAuthenticate`, and others. And, as before, the `global.asax` file can be used to set the values of global or session-level variables and instantiate objects. We look at the use of `global.asax` files in Chapter 6.

ASP+ Application and Session State

One of the useful features in previous versions of ASP that developers were quick to take advantage of was the provision of global and user-level scope for storing values and object instances. This uses the `Application` and `Session` objects in ASP, and these objects are still present in ASP+. Although backwards compatible, however, the new `Application` and `Session` objects offer a host of extra features.

Using Application State

As in previous versions, each ASP application running on a machine has a single instance of the `Application` object, which can be accessed by any pages within that application. Any values or object references remain valid as long as the application is 'alive'. However, when the `global.asax` file is edited, or when the last client `Session` object is destroyed, the `Application` object is also destroyed.

Using the Application object for storing simple values is useful, and there are the usual `Lock` and `Unlock` methods available to prevent users from corrupting values within it through concurrent updates to these values. This can, however, cause blocking to occur while one page waits to access the `Application` object while it is locked by another page.

> *Note that when a page finishes executing or times out, or when an un-handled error occurs, the `Application` object is automatically unlocked for that page.*

Bear in mind the impact of storing large volumes of data in an `Application` object, as this can absorb resources that are required elsewhere. You also need to ensure that any objects you instantiate at `Application`-level scope are thread safe and can handle multiple concurrent accesses. Another limitation in the use of the `Application` object is that it is not maintained across a Web farm where multiple servers handle user requests for the same application, or in a 'Web garden' where the same application runs in multiple processes within a single, multi-processor machine.

Using Session State

While the `Application` object is little changed from earlier versions of ASP, the `Session` object in ASP+ has undergone some quite dramatic updating. It is still compatible with code from earlier versions of ASP, but has several new features that make it even more useful.

The biggest change is that the contents of the `Session` object can now (optionally) be stored externally from the ASP+ process, in a new object called a **Session State Store**. It is managed by a new Windows service called the **State Server Process**, and this persists the content of all users `Session` objects – even if the ASP+ process they are running under fails. It also removes the content of sessions that have terminated through a time-out or after an error.

Alternatively, the `Session` content can be serialized out into a temporary SQL Server database table, where ASP+ talks directly to SQL Server. This means that it can be reloaded after an application or machine failure, providing a far more robust implementation of `Session` object storage than previous versions. It's therefore ideal for applications that require a session-level state management feature, for example a shopping cart.

This new state storage system also has another direct advantage. When using a Web farm to support a large-scale application or Web site it has always been a problem managing sessions, as they are not available across the machines in the Web farm.

The new ASP+ Session State Store can be partitioned across multiple servers, so that each client's state can be maintained irrespective of which server in the Web farm they hit first.

At last, ASP+ also allows session state to be maintained for clients that don't support cookies. This was proposed in earlier versions of ASP, but never materialized. Now, by using a special configuration setting, you can force ASP+ to 'munge' the session ID into the URL. This avoids all of the previously encountered problems of losing state information, for example when the URLs in hyperlinks do not use the same character case as the page names.

Finally, the State Server Process will also be able to expose information about the contents of the Session State Store, which will be useful for performance monitoring and administrative tasks such as setting maximum limits for each client's state. We examine the way that sessions can be managed in Chapter 6.

ASP+ Error Handling, Debugging, and Tracing

Error handling and debugging has long been an area where ASP trailed behind other development environments like Visual Basic and C++. In ASP+, there are several new features that help to alleviate this situation. It's now possible to specify individual error pages for each ASP+ page, using the new `ErrorPage` directive at the start of an ASP+ page:

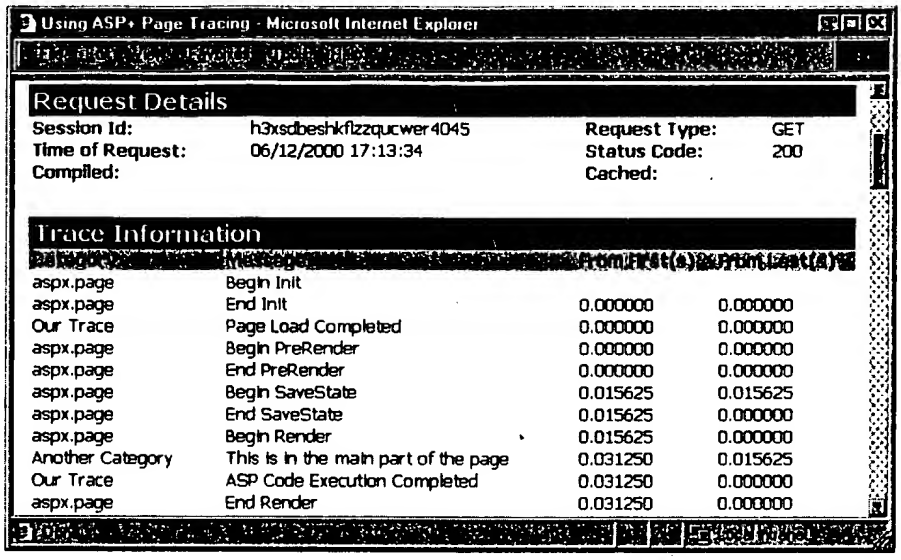
```
<%@Page ErrorPage="/errorpages/thispage.aspx"%>
```

If a 'Not Found', 'Access Forbidden' response is generated, or an 'Internal Server Error' (caused by an ASP code or object error) occurs while loading, parsing, compiling or processing the page, the custom error page you specify is loaded instead. In this page, you can access the error code, the page URL, and the error message. The custom error page, or other error handler, can alternatively be specified in the `config.web` file for an application.

If no error page is specified then ASP+ will load its own error page, which contains far more detail than before about the error and how to rectify it. Settings in the `config.web` file also allow you to specify that this information should only be presented to a browser running on the Web server, and in this case remote users will just receive a simple error indication. Alternatively, you can create a procedure in the ASP+ page that captures the `HandleError` event, and doing so also prevents the default error page from being displayed.

Another welcome new feature is that Visual Basic now supports the `try...catch...finally` error handling technique that has long been the mainstay in languages like C++, and more recently in JScript. More details of this can be found in Appendix B. ASP+ also improves debugging techniques by including a new **Debugger** tool. This is the same debugger that is supplied with Visual Studio, allowing local and remote debugging.

Finally, ASP+ now includes comprehensive tracing facilities, so you no longer need to fill your pages with `Response.Write` statements to figure out what's going on inside the code. Instead, by turning on tracing at the top of the page with the new `Trace` directive, you can write information to the `Trace` object and have it rendered automatically as an HTML table at the end of the page. Tracing can also be enabled for ASP+ applications by adding instructions to the `config.web` file. The trace information automatically includes statistics that show the response time and other useful internal parameters:



On top of this, a Web-based viewer is provided that allows you to examine the contents of the `TraceContext` object's log file. We look at the whole topic of error handling, debugging and tracing in Chapter 4.

Other ASP+ Features

To finish of this brief tour of the new features in ASP+, we'll look at some other topics that fall under the 'general' heading. These include security management, sending e-mail from ASP+ pages, and server-side caching.

New Security Management Features

In Chapters 4 and 6, you'll see how ASP+ implements several new ways to manage security in your pages and applications. As in ASP 3.0, the Basic, Digest and NTLM (Windows NT) authentication methods can be used. These are implemented in ASP+, using the services provided by IIS in earlier versions of ASP. There is also a new authentication technique called **Passport Authentication**, which uses the new Managed Passport Profile API. It's also possible to assign users to roles, and then check that each user has the relevant permission to access resources using the `IsCallerinRole` method.

An alternative method is to use custom form-based authentication. This technique uses tokens stored in cookies to track users, and allows custom login pages to be used instead of the client seeing the standard Windows Login dialog. This provides a similar user experience to that on amazon.com and yahoo.com. Without this feature, you need to write an ISAPI Filter to do this – with ASP+ it becomes trivially simple.

Server-Side Caching

ASP+ uses server-side caching to improve performance in a range of ways. As well as caching the intermediate code for ASP pages and various other objects, ASP has an **output cache** that allows the entire content of a page to be cached and then reused for other clients (if it is suitable).

There is also access to a custom server-side cache, which can be used to hold objects, values, or other content that is required within the application. Your pages and applications can use this cache to improve performance by storing items that are regularly used or which will be required again. The cache is implemented in memory only (it is not durable in the case of a machine failure), and scavenging and management is performed automatically by the operating system.

Getting Started with ASP+

Having seen what ASP+ is all about, and some details of the technologies that support it behind the scenes, it's time to get your hands dirty and build some applications. You can download the sample files for this book to run on your own server, and modify and extend them yourself. But first, if you haven't already done so, you must install ASP+.

The latest version of ASP+ can be downloaded from the Microsoft Web site. At the time of writing, the exact location of the download was unknown, but you can reach it via our support website at http://www.wrox.com/beta. It is also available as a CD for a minimal cost, and is part of Visual Studio 7.

As for tools, at the time of writing we are using the usual ASP developer's friend, Windows NotePad. Of course, you can continue to use Visual InterDev or any other development tool you wish that supports ASP – it just won't be much help with the new object syntax and server-side controls in ASP+. But as long as it doesn't mangle any code that it does not recognize, it will be fine until better tools become available.

And, if like us you're a confirmed 'simple text editor' ASP developer, you might like to try one of the alternatives to Windows NotePad that offers extra features. Our current favorite is TextPad (http://www.textpad.com/).

Installing ASP+

Installing ASP+ is just a matter of running the executable setup file. However, you should ensure that you have installed Internet Explorer version 5.5 first. If not, download it or install it directly from http://msdn.microsoft.com/downloads/webtechnology/ie/iepreview.asp.

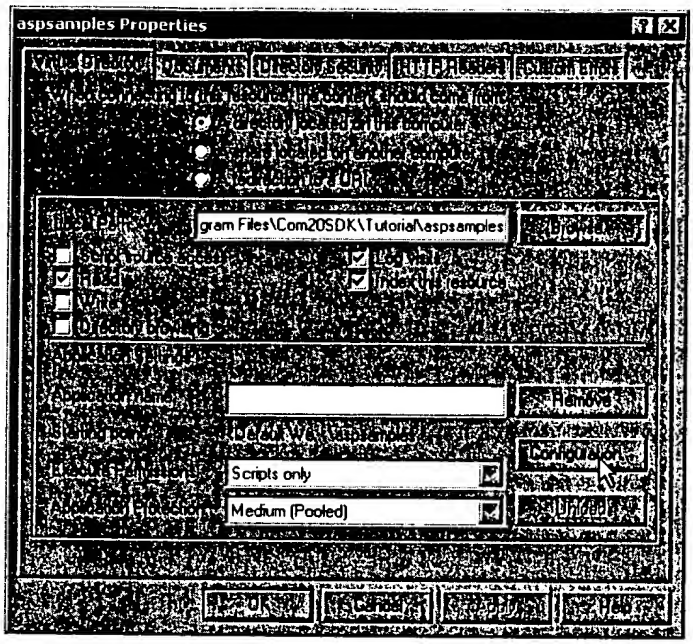
Make sure that you close **all** other applications before installing IE 5.5, as it updates many of the Windows 2000 operating system files. Once installation is complete, you are ready to run ASP+. No other configuration is required at the moment, as the default configuration will do nicely for our first experimental efforts.

Creating an ASP+ Application

In ASP 2.0 and 3.0, it's necessary to take some definite actions to create an ASP application, especially if you want to run any components that the application uses in a separate process. The good news is that, with ASP+, none of this is actually required. And you don't have to register any ASP+ components either.

As we saw earlier, a file named `config.web` controls the configuration of an ASP+ application. It is stored in the root folder of that application. However, there is a default `config.web` file (automatically installed in your `Program Files\COM2OSDK\` folder when you install the runtime) that is used for all ASP+ applications. So, all you have to do to get started is create a subdirectory under your `InetPub\WWWRoot` folder and place your ASP+ pages there.

Of course, you can still create a folder outside the `WWWRoot` directory, and set up a virtual directory to point to it in the Internet Services Manager if required (as in previous versions of ASP). There is no need to set any of the configuration options in the Application Settings section of the Properties dialog for this application, or in the Configuration dialog – the default settings will work fine:

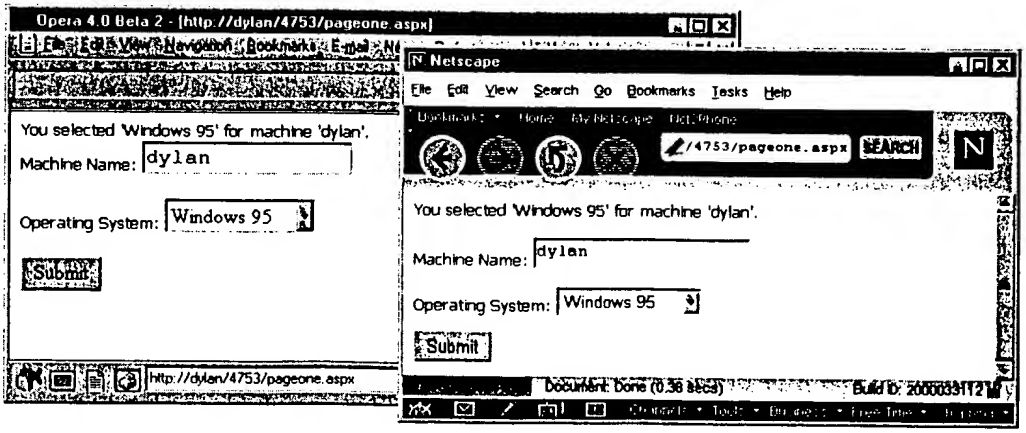


Later, you can add a `config.web` file and a `global.asax` file to the application's root folder if required to specify the configuration settings and application-level event handlers.

Testing Your Installation

Once you've installed the ASP+ runtime framework (and Internet Explorer 5.5 for the preview version of ASP+), you can try it out. An easy way to confirm that it's working is to run one of the sample files we provide. The simple example page named `pageone.aspx` that we looked at earlier is included in the `Chapter01` folder of the samples for this book (available from http://www.wrox.com/).

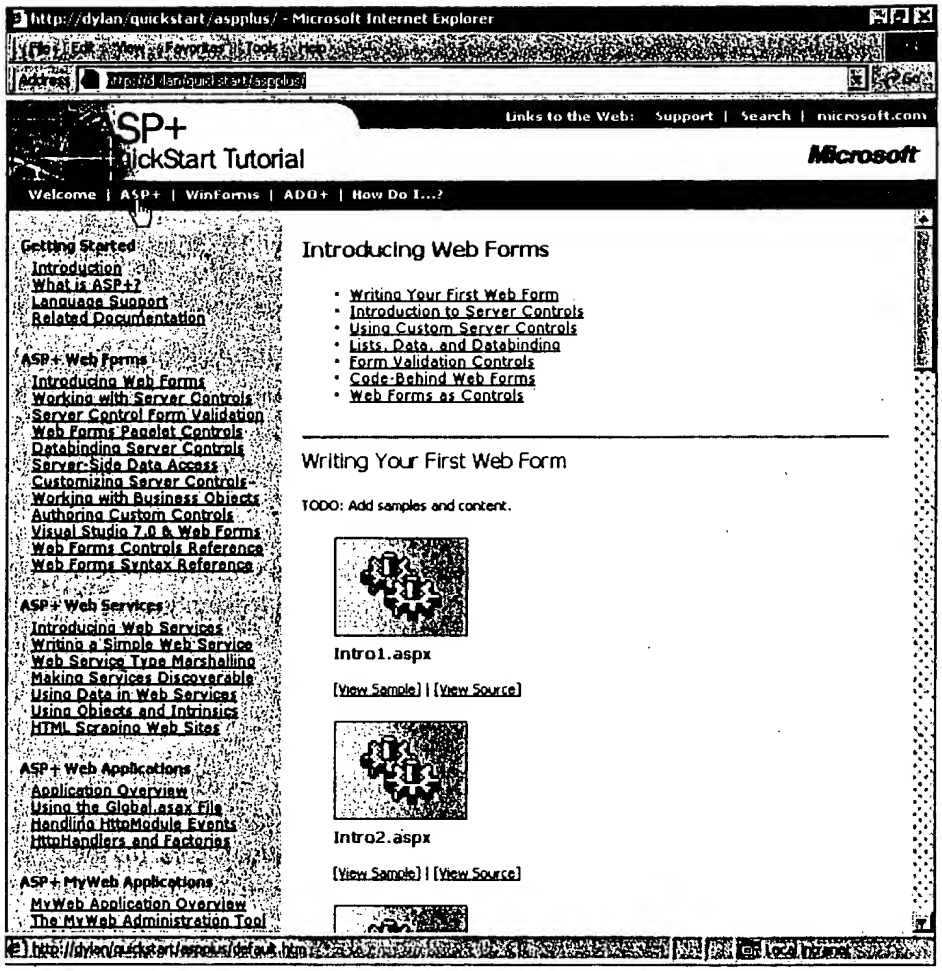
Simply copy it to the `InetPub\WWWRoot` directory on your server and open it from a browser using the URL http://localhost/pageone.aspx or http://your_server_name/pageone.aspx. You should get this:



We've used Netscape Navigator 6 and Opera 4 here to prove that the page doesn't depend on the unique capabilities of Internet Explorer.

If the page doesn't work, check out the 'read me' text file that comes with ASP+ for late-breaking information. Alternatively, have a look at the SDK documentation provided with ASP+, or available at the Microsoft Web site, to see a full description and the remedy for any error message that you get.

Once you are up and running, the next step is to take a look at the Quick Start tutorials. There are examples of all kinds of ASP+ pages, Web services, and applications that you can try out and view the source code. Open the samples from http://localhost/quickstart/ or http://machinename/quickstart/:



About the ASP+ Final Release

Obviously, the preview version of ASP+ and the runtime framework that we are using is not absolutely complete. However, it is classed as being 'feature complete', which means that only minor changes and additions are expected between now and the final release. In this last section, we'll examine some of the things that you can expect to see in the final release that are not available, or that aren't yet working properly.

Multiple Windows Platform Support

The final version of the NGWS framework and ASP+ is aimed at all of the current and recent Windows platforms, including Windows 2000, Windows NT4, Windows 95 and Windows 98. The preview release, however, is only designed for use on Windows 2000 Server and Windows 2000 Professional. The versions for Windows 95 and Windows 98 will be limited-functionality 'personal' versions, but will allow these operating systems to provide a local source for the execution of ASP+ pages. This will be useful for building applications designed for running locally.

XHTML Compliance

At the moment, the output generated by the server-side ASP+ controls is basic HTML 3.2, and is not XHTML compliant. Good coding practice suggests that all Web pages should be compliant with the new XHTML recommendations from the World Wide Web Consortium (W3C), so as to allow them to be manipulated if required by an XML parser or other application that expects content to be well-formed in XML terms.

A complete specification of XHTML version 1.0 can be obtained from the W3C Web site at http://www.w3.org/TR/xhtml1, and Microsoft will attempt to generate XHTML-compliant HTML code from server-side components in the final release of ASP+. However, as some popular browsers can behave oddly when confronted with XHTML, the final level of support is difficult to judge at the moment.

Client-Specific Output Formats

Most of the intelligent server-side controls supplied in the preview version of ASP+ only output standard HTML 3.2. However, some (such as the validation controls we look at in Chapter 4) do detect Internet Explorer 4 and above, and generate output that takes advantage of the DHTML capabilities of this browser. This provides better performance and a better user experience, as it dramatically reduces the need for round-trips to the server each time the user changes the selected data in the control.

In the later beta and release versions of ASP+, there will be more controls of this type. There will also be controls aimed at creating output in different formats entirely, for example Wireless Markup Language (WML). This might be a separate set of controls in some cases; however, due to the extreme incompatibilities between the user interfaces and client capabilities for these types of Internet device.

New Administration Tools

Finally, the release version of ASP+ will include administration tools allowing you to configure and maintain applications more easily. You can expect to see tools to manage the `config.web` configuration files and `global.asax` application files. There should also be graphical interfaces for viewing application performance, and examining detailed trace information while debugging complete applications.

Summary

In this chapter, we've attempted to provide a complete overview of what is new and what has changed in ASP+, compared to earlier versions of ASP. ASP+ is the new generation of Microsoft's successful Active Server Pages technology, and represents a real advance in ease of use and power.

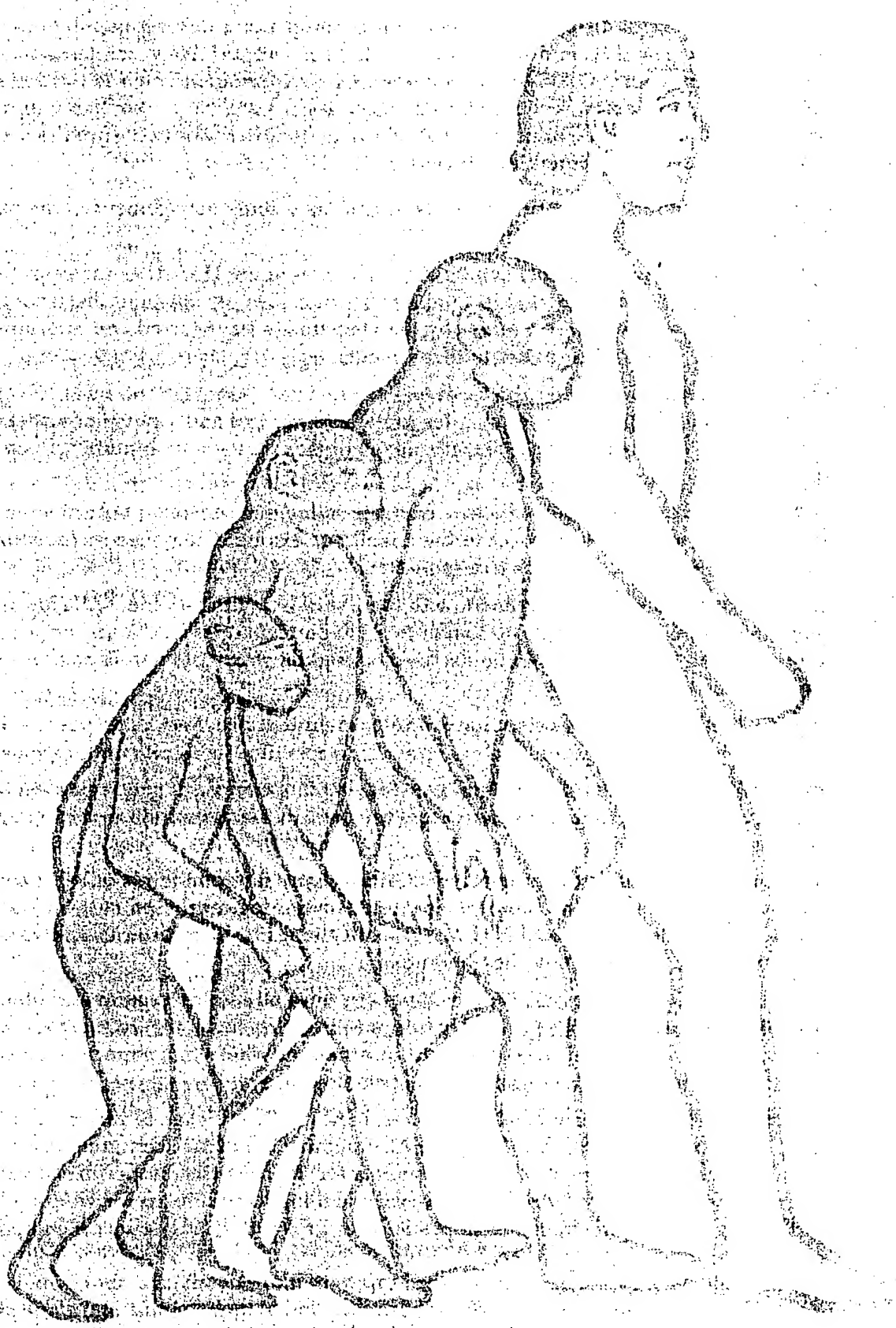
ASP+ is designed to remove many of the existing limitations of ASP, such as the dependence on script languages, poor support for object-oriented design and programming techniques, and the need to continuously re-invent techniques for each page or application you build. Instead, ASP+ combines the ability to build more complex and powerful applications, with a reduced requirement for the developer to write repetitive code. For example, the process of maintaining values in HTML form controls and posting these values back to the server ('round tripping') requires quite a lot of code to maintain the state within the page. ASP+ does all this work for you automatically.

At the same time, the world out there is changing. The proportion of users that will access your site through an 'Internet device' such as a mobile cellular phone, personal digital assistant (PDA), TV set-top box, games console, or other device will soon be greater that the number using a PC and a traditional Web browser. ASP+ provides solutions that help to reduce the work required for coping with these disparate types of client.

The rapidly changing nature of distributed applications requires faster development, more componentization and re-usability, and wider general platform support. New standards such as the Simple Object Access Protocol (SOAP) and new commercial requirements such as business-to-business (B2B) data interchange require new techniques to be used to generate output and communicate with other systems. To meet all these requirements, ASP has been totally revamped from the ground up into a whole new programming environment that includes:

- **Pages** that use the new server-side controls to automate state management in the page, and reduce the code you have to write.
- **HTML Server-side Controls** that can be used to generate the HTML elements in the page output, and allow code to be used to set the properties (i.e. attributes) of these controls at runtime. They also allow events raised by the elements to be detected and appropriate code executed on the server in response to these events.
- **Rich Controls** that run on the server can be used to create more complex HTML elements and objects in the page output. ASP+ includes a calendar control and a range of grid, table and list controls. These controls can also take advantage of server-side data binding to populate them with values.
- **Web Services** that allow developers to create pages that are generally not rendered as visible output, but instead provide services to the client. For example, they can include functions that return specific values in response to a request.
- Better **Configuration and Deployment**, with the use of human-readable XML-format configuration files. Components no longer need to be registered on the server (using `regsvr32`), and applications can be deployed using file copy commands, the FrontPage server extensions, or FTP.
- Extended **Application and Session State Management** that provides a persistent and more scalable environment for storing values relevant to specific clients and applications.
- Improved **Error Handling, Debugging, and Tracing** features. Each page can have its own 'error page', and can also display values that are used in the page code as it executes, providing a 'trace' facility.
- New **Security Management Features**, which allow many different kinds of login and user authentication to be used. Instead of the default browser login prompt, custom login pages can be used with Windows 2000 and NTLM authentication. It is also easier to manage users depending on the role or group they belong to.
- **Custom Server-side Caching** allows developers to store all kinds of values and objects locally on the server for use in ASP+ pages. The runtime can also cache the *output* from ASP+ pages. This can provide a huge performance boost in situations where a dynamically created page is the same for many visitors, such as a product catalog.
- **A Range of Useful Components** are shipped with ASP+. These class libraries can help to make writing Web applications easier. Examples include: the 'SendMail' component, encryption/decryption components, components for defining custom performance counters, components for reading and writing to the NT event log, components for working with MSMQ, network access components (replacements for 'WinInet'), data access components, etc.

In the remainder of this book, we'll examine all thee topics in more detail, and show you how you can use ASP+ to build powerful and interactive Web-based distributed applications more quickly and efficiently than ever before.



2

ASP+ Pages

In the previous chapter we examined the problems with the existing ASP architecture, and how ASP+ has been designed to overcome some of these. Obviously not every problem with designing Web applications is cured by ASP+, but the new environment is such a leap forward in many ways, that many of the problems have been solved.

We also showed you how a simple ASP page could be converted to ASP+ to make use of some of the new built in features, such as control state, but now it's time to look at ASP+ pages in more detail. So, in this chapter we are going to look at:

- The way ASP+ pages can make your code cleaner
- How to program against the ASP+ Object Model
- What Web Controls are, and how they can be used
- How the different families of Web Controls target different uses
- How the Web Controls have an Object Model

By the end of the chapter you'll see how great ASP+ is as a development environment, and the rich set of facilities it has to offer.

Coding Issues

In the previous chapter we looked at two of the main disadvantages of existing ASP applications – those of state management, and the coding model. ASP+ introduces a new way of coding, and for those of you who have programmed in event driven languages such as Visual Basic or DHTML scripters, then this model will seem familiar, although it might seem a little alien to programmers who have only coded in script languages in ASP. However, it's extremely simple, providing many advantages, and leads to much more structured code. No longer is the code intermixed with the HTML, and the event driven nature means the code is easily separated into related blocks.

We previously saw how this worked for an `OnServerClick` event on a server control, and how the code to be run was broken out into a separate event procedure. That example showed a list of operating systems in a SELECT element. Using lists like this is extremely common, and often the list of items is generated from a data store. So, let's look at the difference between an ASP page and a new ASP+ page to do the same thing.

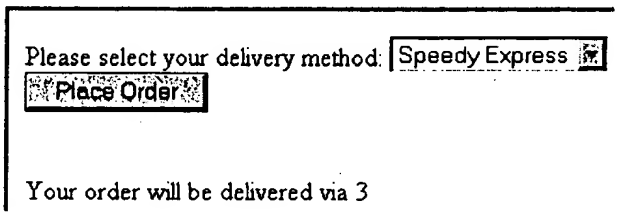
Coding the Old Way

To produce a list of items from a data store in ASP you have to loop through the list of items, manually creating the HTML. When the item is selected, the form is then posted back to the server. Your code can look something like this:

```
<html>

<form action="OldAsp.asp" method="post">

  Please select your delivery method:
  <select id="ShipMethod" Name="ShipMethod" size="1">
<%
  Dim rsShip
  Dim SQL
  Dim ConnStr
  Dim ItemField

  Set rsShip = Server.CreateObject("ADODB.Recordset")

  SQL = "select * from Shipping_Methods"
  ConnStr = "Driver={SQL Server}; Server=localhost; " & _
            "Database=AdvWorks; UID=sa"

  rsShip.Open SQL, ConnStr

  While Not rsShip.EOF
    Response.Write "<option value='" & _
                   rsShip("ShippingMethodID") & "'>" & _
                   rsShip("ShippingMethod") & "</option>"
    rsShip.MoveNext
  Wend

  rsShip.Close
  Set rsShip = Nothing
%>
  </select>
```

```
  <br>
  <input type="submit" value="Place Order">
</form>

<%
  If Request.Form("ShipMethod") <> "" Then
    Response.write "<br>Your order will be delivered via " & _
                   Request.Form("ShipMethod")
  End If
%>
</html>
```

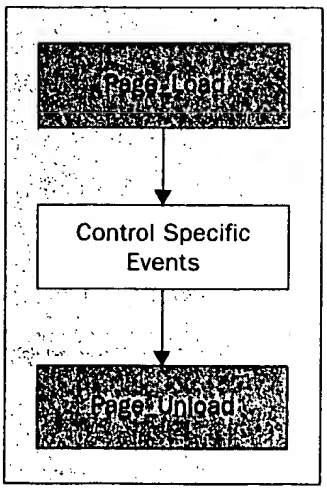
There's nothing very complex about this code – it loops through a Recordset building a select list, and allows the user to select a value and submit that back to the server. In many situations you'd probably build some sort of database query or update based upon the selected value, but in our example we are printing out the selected value:



Notice that only the ID is available in the submitted page. That's fine if we're going to use it in some form of data query, but not if we need to display it again.

Coding in ASP+ Pages

Before we look at how the above ASP page would be coded in ASP+, let's start by examining the event order. In ASP+ we have a structured set of events, so our code can be placed in these, rather than interspersed with the HTML. The event order is:



The `Page_Load` event is always the first event, and is fired **every** time the page is loaded. After that any control specific events that are implemented are fired, and finally the `Page_Unload` is **always** fired last.

Since our Web controls can be accessed server-side, this means we can put the loading of data into them in the `Page_Load` event.

For example, rewriting the original ASP page in ASP+ we get:

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>

<html>
<head>

<script language="VB" runat="server">

  Sub Page_Load(Source As Object, E As EventArgs)

      Dim myCommand As ADOCommand
      Dim myReader  As ADODataReader
      Dim SQL       As String
      Dim ConnStr   As String

      SQL = "select * from Shipping_Methods"
      ConnStr = "Provider=SQLOLEDB; Data Source=(local);_
                 Initial Catalog=AdvWorks; User ID=sa"

      myCommand = New ADOCommand(SQL, ConnStr)
      myCommand.ActiveConnection.Open()

      myCommand.Execute(myReader)

      While myReader.Read()
        ShipMethod.Items.Add(New ListItem(myReader.Item("ShippingMethod"), _
                             myReader.Item("ShippingMethodID")))
      End While

  End Sub

  Sub PlaceOrder_Click(Source As Object, E As EventArgs)

    YouSelected.Text = "Your order will be delivered via " & _
                       ShipMethod.SelectedItem.Text

  End Sub

</script>

<form runat="server">

  Please select your delivery method:

  <asp:DropDownList id="ShipMethod"
       runat="server"/>

  <br/>

  <asp:button id="PlaceOrder" Text="Place Order"
       onclick="PlaceOrder_Click"
       runat="server"/>
```

```
  <br/>

  <asp:Label id="YouSelected" runat="server"/>
</form>
</html>
```

Let's look at this code in detail to see what the changes are and why they've been done, starting with the HTML form:

```
<form runat="server">

  Please select your delivery method:

  <asp:DropDownList id="ShipMethod"
       runat="server"/>

  <br/>

  <asp:button id="PlaceOrder" Text="Place Order"
       onclick="PlaceOrder_Click"
       runat="server"/>

  <br/>

  <asp:Label id="YouSelected" runat="server"/>
</form>
```

This has three ASP+ Web controls – a drop down list, and button, and a label. We'll explain more about these throughout the chapter, but for now we want to concentrate on the event side of things. So, just remember that these are server-side controls, and can therefore be accessed from within our server-side code. Now let's look at the coding bits in more detail.

One important point to note is that for server controls to take part in postback, they must be within a server-side form control. For example:

```
<form runat="server">

  ' controls go here

</form>
```

At the start of the page we place the Import statements, to tell ASP+ which code libraries we wish to use. The two below are for data access, and we'll explain more about these in the next chapter:

```
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.ADO" %>

<html>
<head>
```

The Page_Load Event

Next we start the script block, and define the Page_Load event. Remember this will be run every time the page is loaded.

```
<script language="VB" runat="server">

  Sub Page_Load(Source As Object, E As EventArgs)
```

So, within this event we want to query the database and build our list. We're not going to explain the following data access code, since it's covered in the next chapter, but it's roughly equivalent to the code in the previous example, simply creating a set of records.

```
Dim myCommand As ADOCommand
Dim myReader  As ADODataReader
Dim SQL       As String
Dim ConnStr   As String

SQL = "select * from Shipping_Methods"
ConnStr = "Provider=SQLOLEDB; Data Source=(local);_
           Initial Catalog=AdvWorks; User ID=sa"

myCommand = New ADOCommand(SQL, ConnStr)
myCommand.ActiveConnection.Open()

myCommand.Execute(myReader)
```

Once the records are created, we add them to the list (ShipMethod). In the ASP example we actually had to create the HTML OPTION elements in the loop, but in the ASP+ page we can just use the Add method of the list to add items. We showed earlier that the list is declared to run server-side, so we can just call its methods and properties server-side. This is more like the Visual Basic programming environment, where we're dealing with controls.

```
While myReader.Read()
  ShipMethod.Items.Add(New ListItem(myReader.Item("ShippingMethod"), _
                       myReader.Item("ShippingMethodID")))
End While
```

At this stage the code in the Page_Load event has finished. On the first load there are no control events to run, so processing will continue with the Page_Unload event, if implemented.

Web Control Events

When we select an item in the list and click the button, we invoke the postback mechanism. Our button was defined as:

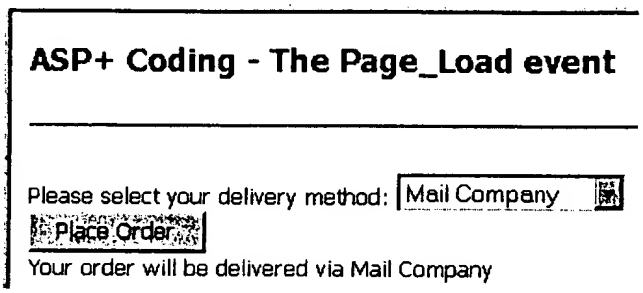
```
<asp:button id="PlaceOrder" Text="Place Order"
    onclick="PlaceOrder_Click"
    runat="server"/>
```

The onclick property identifies the name of the procedure to be run when the button is clicked. Remember that this is server-side event processing, so there is no special client-side code.

When this button is clicked the form is submitted back to itself, and the defined event handler is run:

```
Sub PlaceOrder_Click(Source As Object, E As EventArgs)

  YouSelected.Text = "Your order will be delivered via " & _
                     ShipMethod.SelectedItem.Text

End Sub
```

This just sets the text of a label to the value selected:



This shows one interesting point. Because the list control is a server control we have access not only to the value, but also to the text of the selected item. This wasn't possible (at least not easily) with the ASP page.

So, let's just reiterate these points:

- Server based controls can be accessed from server code.
- The `Page_Load` event is run every time the page is loaded.
- The control event is only run when fired by a server control.

There is, however, one big flaw with our code above. Because the `Page_Load` runs every time the page loads, the code in it will be run even under a postback scenario. That means we are performing the data query and filling the list every time, so whatever the user selected would be overwritten (as well as being unnecessary).

The Page.IsPostBack Property

The `Page.IsPostBack` property is designed to counter this problem, as it is a Boolean value that is set to `True` whenever the page has been posted to (for example, when running the event procedure for a control). We can use this property in the `Page_Load` event so that our data access code is only run the first time the page is loaded:

```
Sub Page_Load(Source As Object, E As EventArgs)

  If Not Page.IsPostBack Then

    Dim myCommand As ADOCommand
    Dim myReader  As ADODataReader
    Dim SQL       As String
    Dim ConnStr   As String

    SQL = "select * from Shipping_Methods"
    ConnStr = "server=localhost;uid=sa;pwd=;database=AdvWorks"
```

```
        myCommand = New ADOCommand(SQL, ConnStr)
        myCommand.ActiveConnection.Open()

        myCommand.Execute(myReader)

        While myReader.Read()
          ShipMethod.Items.Add(New ListItem(myReader.Item("ShippingMethod"),_
                               myReader.Item("ShippingMethodID")))
        End While
      End If

    End Sub
```

The Page ViewState

We've now ensured that this code runs only when the page is first loaded. We don't have to worry about the contents of the list disappearing because the contents are held within the ViewState. We won't show it here as the contents aren't reader friendly, but this contains everything about the current state of the control.

The ViewState means that you don't have to do anything at all to preserve the contents of controls during postback.

Web Controls

In the previous chapter when we looked at the improvements in the code structure, we showed code responding to an event using the HTML SELECT element. These standard HTML controls are all very well, and have been designed to map one to one with their HTML equivalents. And this is one of their problems, since you do not automatically get the benefits of the uplevel ASP+ controls. For example, consider the background color. For a SPAN you set the CSS `style` property, and on a table you set the `BGColor` property. The ASP+ controls provide a consistent model, whereby you can just set the `BackColor` property on all controls, and the correct HTML is automatically rendered.

There are four sets of Web Controls:

- Intrinsic Controls, which map to simple HTML elements
- List Controls, to provide data flow across a page
- Rich Controls, to provide rich UI and functionality
- Validation Controls, to provide a variety of data validation

As we go through this chapter you'll see examples from all of these families, and some will be covered in more detail in the next chapter when we look at data and data binding.

HTML Controls or ASP+ Controls

At this stage you might be questioning the actual need for HTML controls that work server-side, as well as ASP+ controls. That's a question that the ASP+ team have asked themselves several times, but they opted for the flexible approach, allowing the user to decide.

For example, consider the following from Chapter 1:

```
<input type="text" id="txtName" runat="server">
```

This uses the HTML control, but includes the `runat="server"` attribute to make it available for server-side programming. How does this differ from the following?

```
<asp:TextBox id="txtName" runat="server"/>
```

Well, once it's rendered it doesn't differ at all. The first produces the following HTML:

```
<INPUT name="txtName" id="txtName" type="text">
```

And the second produces this HTML:

```
<input name="txtName" type="text" id="txtName">
```

So, if there's no difference, which should you use? Well, as one member of the ASP+ team puts it "it's a lifestyle choice". Use whichever set of controls you feel happier with. The HTML controls keep you closer to the actual content, but they don't offer the uplevel functionality (such as consistent naming and so on). On the other hand, the ASP+ controls provide a more consistent programming model, but divorce you a little from the actual content output. Through the rest of this book we'll be using the ASP+ controls.

Using Web Controls

You use Web Controls in the same way as you use HTML controls. The only difference is that they must have the `runat="server"` attribute set. You don't have to do anything special to access this code library, as it's available by default, but you do have do ensure you use the correct tag prefix (or code namespace as it is sometimes called) when using the control. For example, a normal HTML list control is added to the page like this:

```
<select id="ShipMethod">
```

An ASP+ list control has the following form:

```
<asp:ListBox id="ShipMethod">
```

The code namespace is the `asp:` at the beginning of the control name. The reason we have the tag prefix, is to ensure that controls with the same names can be uniquely identified. This might not seem a big problem, but when you realize you can author your own controls, you can see what a problem this might be. We suspect there will also be a large third party market for controls, so it's extremely important that they are named correctly.

The Object Model

An important point to remember is that Web Controls (and all controls for that matter) are objects, and are fully supported by all of the usual things that objects have: methods, properties and events.

Properties and Methods

Once a Web Control has been added to an ASP+ page, you can set properties and call methods at will. You've already seen one example of this – the `DropDownList`:

```
<script language="vb" runat="server">

  Sub Page_Load(Source As Object, E As EventArgs)

    . . .

      ShipMethod.Items.Add(
                  New ListItem(myReader.Item("ShippingMethod"), _
                               myReader.Item("ShippingMethodID")))

  End Sub

  Sub PlaceOrder_Click(Source As Object, E As EventArgs)

    YouSelected.Text = "Your order will be delivered via " & _
                       ShipMethod.SelectedItem.Text

  End Sub

</script>

  <asp:DropDownList id="ShipMethod"
```

In the `Page_Load` event we are calling the `Add` method of the `ShipMethod` control (a listbox), to programmatically add items to the list. In the `PlaceOrder_Click` procedure we are setting the `Text` property of a label. For anyone who has worked in DHTML or Visual Basic this isn't anything new, but it is a big leap forward for ASP server-side code.

Responding to Events

In the previous chapter you saw that the new object model allows us to respond to client-side events within server-side code. If you've done any client-side programming you might well be used to that, but this new model allows for cross-browser compatibility, by rendering pure HTML to the client, and responding to events on the server.

The use of server-side events is the same as for client event processing – you just specify the event name, and then the event procedure to be run when the event is fired:

```
<asp:Button id="FooBtn" onclick="FooBtn_onclick">
```

You then implement the event procedure as follows (in VB):

```
Sub FooBtn_onclick(Sender As object, E As EventArgs)
```

In C# the procedure declaration would be:

```
void FooBtn_onclick(Object Sender, EventArgs E)
```

The most common event you'll be responding to will be the `OnClick` event, although the controls do support other events. For a list of these you need to consult the documentation for the particular control.

Control Families

Now that we've described the reasons for using Web Controls, let's look at the control families in detail. Remember that there are four of them:

- Intrinsic Controls, which map to simple HTML elements
- List Controls, to provide data flow across a page
- Rich Controls, to provide rich UI and functionality
- Validation Controls, to provide a variety of data validation

In this chapter we'll be covering the Intrinsic Controls and the Rich Controls. The List controls are covered in Chapter 3, when we look at data and data binding, and the Validation Controls are covered in Chapter 4, when we look at advanced ASP+ Page features.

Intrinsic Controls

The Intrinsic Controls are designed to provide replacements for the standard set of HTML controls. So, why do we need replacements, especially since the existing HTML controls can be run server-side? Simply put, it's consistency. One of the problems with HTML has been its lack of consistent naming for similar controls. Consider the case of an inputting data, as there are many forms this can take:

```
<input type="radio">
<input type="checkbox">
<input type="button">
```

Although these are all input controls, they all behave in a different manner, and should really be different controls, such as:

```
<asp:RadioButton>
<asp:CheckBox>
<asp:Button>
```

Text input in HTML is the opposite of this, with two controls performing the same task:

```
<input type="text">
<textarea>
```

Both of these provide areas for text input – the first only allows a single line, while the second allows multiple lines. A more sensible solution is a single control, where you can specify the number of lines:

```
<asp:TextBox rows="1">
<asp:TextBox rows="5">
```

As well as having a set of properties relating specifically to the control, the Web Controls also accept standard HTML properties. That's because these controls actually render HTML on the client, so they map HTML properties through to the rendered HTML.

We'll look at these controls in more detail, taking them in related groups.

Control Transfer

There are four types of controls that allow passing of control back to the server:

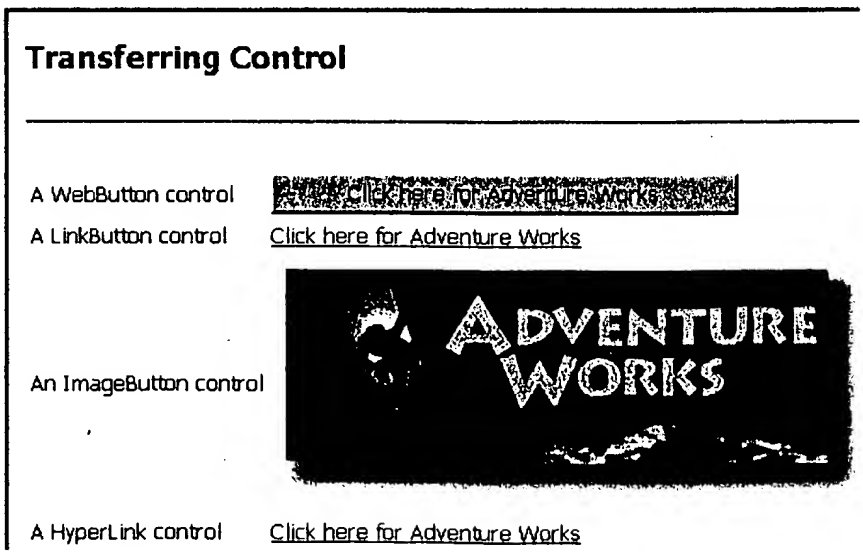
- `Button`, which acts as a standard submit button. Use this when you want to perform normal postback to the server.
- `LinkButton`, which allows custom processing before postback is actually received on the server. This allows you to have a button and perform server-side scripting instead of the normal postback routine. It is the only intrinsic control that relies on client-side scripting.
- `ImageButton`, which is a standard image button. Use this when you want to display an image that performs postback to the server.
- `Hyperlink`, which performs the actions of an a tag.

This gives us a flexible set of ways of performing postback, and the use of each control is similar:

```
<html>
<form runat="server">

  <table border="0">
    <tr>
      <td>A WebButton control</td>
      <td><asp:Button id="AdvWorksWebBtn"
                Text="Click here for Adventure Works"
                onclick="AdvWorksWebBtn_Click"
                runat="server"/></td>
    </tr>
    <tr>
      <td>A LinkButton control</td>
      <td><asp:LinkButton id="AdvWorksLinkBtn"
                Text="Click here for Adventure Works"
                onclick="AdvWorksLinkBtn_Click"
                runat="server"/></td>
    </tr>
    <tr>
      <td>An ImageButton control</td>
      <td><asp:ImageButton id="AdvWorksImg"
                ImageURL="/advworks/images/Frontartcomp.jpg"
                OnClick="AdvWorksImg_Click"
                runat="server"/></td>
    </tr>
    <tr>
      <td>A HyperLink control</td>
      <td><asp:HyperLink id="AdvWorks"
                NavigateURL="/advworks"
                Text="Click here for Adventure Works"
                runat="server" /></td>
    </tr>
```

```
    </table>

  </form>

  <script language="vb" runat="server">

    Sub AdvWorksImg_Click(Sender As Object, _
                          E As ImageClickEventArgs)

      Page.Navigate ("//localhost/advworks")

    End Sub

    Sub AdvWorksWebBtn_Click(Sender As Object, E As EventArgs)

      Page.Navigate ("//localhost/advworks")

    End Sub

    Sub AdvWorksLinkBtn_Click(Sender As Object, E As EventArgs)

      Page.Navigate ("//localhost/advworks")

    End Sub

  </script>
  </html>
```

In all cases I've decided to navigate to another page, but since this page runs on the server, you can perform any processing you like. The output of this code would look like this:



They all achieve the same purpose in this example, but you see the difference when you look at the generated HTML:

The Button control generates a normal submit button:

```
<input type="submit" name="AdvWorksWebBtn"
       value="Click here for Adventure Works"
       id="AdvWorksWebBtn">
```

The LinkButton generates an a tag, but actually posts back to the server:

```
<a id="AdvWorksLinkBtn"
   href="javascript:__doPostBack('AdvWorksLinkBtn', '')">
   Click here for Adventure Works
   </a>
```

The ImageButton generates a normal image input control:

```
<input type="image" name="AdvWorksImg" id="AdvWorksImg"
       src="/advworks/images/Frontartcomp.jpg" border="0">
```

The HyperLink generates a standard a tag that performs normal redirection:

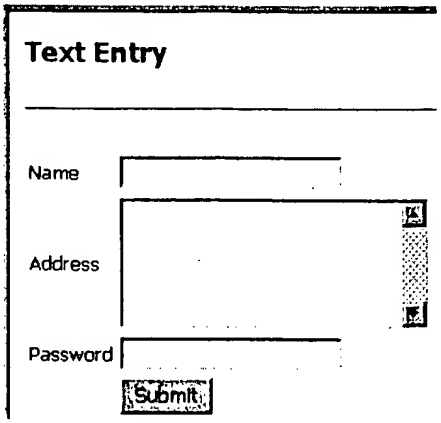
```
<a id="AdvWorks" href="/advworks">
   Click here for Adventure Works</a></td>
```

Which you use depends on how you want your page to look and the processing you need to achieve. The general advice is:

- Hyperlinks don't postback to the server. They directly navigate from one page to another, and are the most efficient way to navigate. However, they assume that no values typed on the origin page need to be accessed or saved before the navigation takes place.
- Buttons and ImageButtons enable you to postback, with the server code having free reign on all client values. You **could** then do a navigation from within one of these event handlers – you could also just as easily update the page and then have it automatically redisplay back to the client.
- LinkButton looks like a hyperlink on the client, but actually causes a postback to the server when clicked. This is the one control we'll look at that requires client-side javascript support in order for it to work. If you don't want to require client-side script support you might want to consider using an ImageButton or Button instead.

Text Entry

The TextBox is the only text entry control, since it not only handles both single and multi line entry, but also password entry too. For example, imagine this form:



This is created with the following:

```
<html>
<head>

<script language="vb" runat="server">

  Sub SubmitBtn_Click(Sender As Object, E As EventArgs)

    YouEntered.Text = "Hi " & Name.Text & _
                      ". You live at " & Address.Text & _
                      " and your password is " & Password.Text

  End Sub

</script>

<form runat="server">

  <table border="0">
    <tr>
      <td>Name</td>
      <td><asp:TextBox id="Name" runat="server" /></td>
    </tr>
    <tr>
      <td>Address</td>
      <td><asp:TextBox id="Address" TextMode="MultiLine"
                Rows="5" Columns="30" runat="server" /></td>
    </tr>
    <tr>
      <td>Password</td>
      <td><asp:TextBox id="Password" TextMode="Password"
                runat="server" /></td>
    </tr>
    <tr>
      <td></td>
      <td><asp:Button id="SubmitDetailsBtn" text="Submit"
                onclick="SubmitBtn_Click" runat="server" /></td>
    </tr>
  </table>

  <p><asp:Label id="YouEntered" runat="server" /></p>

</form>
```

It's pretty obvious stuff. Although the three text entry controls are all `TextBox` controls, they use different modes. The first uses the default of single line, whereas the second use `MultiLine` mode and specifies the size of the text box. The third specifies the mode to be `Password`, so whatever is typed in is not echoed back to the screen. This sort of form is typical for collecting user information from people visiting your site. In our event procedure for the button we just display the details back to the user, but this could just as easily update a database.

Selections

Selections are catered for by the following controls:

- `CheckBox`, which allows multiple options to be selected.
- `RadioButton`, which allows only a single button to be selected.
- `ListBox`, which provides a list of items, allowing single or multiple selection.
- `DropDownList`, which provides a drop down list allowing only a single selection.

Since these give different options, we'll examine them separately.

CheckBoxes

`CheckBoxes` are analogous to the HTML input type checkbox, and have only four properties:

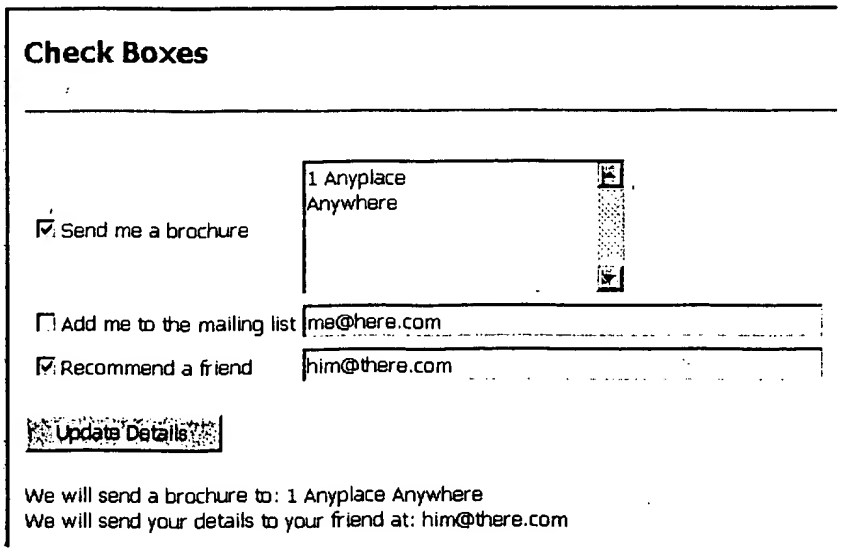
- `Checked`, to indicate whether or not the check box is selected.
- `Text`, which is the text to display.
- `TextAlign`, to indicate whether the text appears to the left or right of the check box. This can be the value `Left` or `Right`, and defaults to `Left`.
- `AutoPostBack`, to indicate whether or not changing the state of the check box automatically posts back to the server.

For example, consider the following code:

```
<html>
<head>

<script language="vb" runat="server">

  Sub UpdateDetails_Click(Sender As Object, E As EventArgs)

    Dim Details As String

    If BrochureChk.Checked Then
      Details = "We will send a brochure to: " & Address.Text & "<br/>"
    End If

    If MailChk.Checked Then
      Details &= "You will be added to out mailing list, " & _
                 "using this email address:" & _
                 EmailAddress.Text & "<br/>"
    End If

    If FriendChk.Checked Then
      Details &= "We will send your details to your friend at: " & _
                 FriendEmailAddress.Text
    End If

    SelectionDetails.Text = Details

  End Sub
```

```
</script>

<form runat="server">

  <table border="0">
    <tr>
      <td><asp:CheckBox id="BrochureChk" Text="Send me a brochure"
                runat="server" /></td>
      <td><asp:TextBox id="Address" TextMode="MultiLine"
                Rows="5" Columns="30" runat="server"/></td>
    </tr>
      <td><asp:CheckBox id="MailChk" Text="Add me to the mailing list"
                checked="true" runat="server" /></td>
      <td><asp:TextBox id="EmailAddress" columns="50" runat="server"/></td>
    </tr>
      <td><asp:CheckBox id="FriendChk" Text="Recommend a friend"
                runat="server"/></td>
      <td><asp:TextBox id="FriendEmailAddress" columns="50"
                runat="server"/></td>
  </table>

  <br/>

  <asp:Button id="UpdateDetails" Text="Update Details"
       onclick="UpdateDetails_Click"
       runat="server"/>

  <p>
    <asp:Label id="SelectionDetails" runat="server"/>
  </p>

</form>
</html>
```

This contains three check boxes, along with three text boxes, a command button and a label. When the button is clicked, its event procedure sees whether each check box is selected, and if so, sets the string to be output in the label:



AutoPostBack

If you want a more immediate return to the server, you can set the `AutoPostBack` property of the check box and set an associated event procedure:

```
<asp:CheckBox id="BrochureChk" Text="Send me a brochure"
     OnCheckedChanged="BrochureChk_Changed"
     AutoPostBack="true" runat="server" />
```

This means that every time the check box is selected or cleared, a postback will occur. Although this gives a more immediate response, it's not useful for all situations, and should be used carefully, since it could make your application appear slower, since postbacks happen more frequently.

RadioButtons

Radio buttons are similar to check boxes, with the addition of allowing only one item in a group to be selected. For this you set the `GroupName` property of the radio buttons to be the same. For example, imagine a list of suitable delivery methods for an order page:

```
<asp:RadioButton ID="Standard" GroupName="Delivery"
     Text="Standard - 7 to 10 days" Checked="true"
     runat="server" />
<br />
<asp:RadioButton ID="Medium" GroupName="Delivery"
     Text="Medium - 3 to 7 days" runat="server" />
<br />
<asp:RadioButton ID="Fast" GroupName="Delivery"
     Text="Fast - Next working day" runat="server" />
```

You could then add the order button and its associated event procedure:

```
<asp:Button id="ProcessOrder" onclick="ProcessOrderBtn_Click"
     Text="Process Order" runat="server" />

  Sub ProcessOrderBtn_Click(Sender As Object, E As EventArgs)

    Dim Message As String

    Message = "Thank you for you order. It will arrive in "

    If Standard.Checked Then
      Message += "7 to 10 days."
    Elseif Medium.Checked Then
      Message += "3 to 7 day."
    Else
      Message += " tomorrow."
    End If

    ThankYou.Text = Message

  End Sub
```

Within the event procedure you only need to test the `Checked` property of each radio button.

ListBoxes and DropDownLists

There are two differences between list boxes and drop down lists. The first is that a drop down list only shows one item at a time. Leading from this is that the drop down list can only have one item selected, whereas the list box can have more than one item selected. This is controlled by the `SelectionMode` property, which can be `Single` for only a single item, or `Multiple` for multiple item selection.

There are three ways of getting the items into these lists. You've seen the first way, by use of the `Add` method, where you programmatically add items to the list. The second is by way of the `ListItem` control:

```
<asp:ListBox id="ListBox" runat="server">
  <asp:ListItem selected="true">Item One</asp:ListItem>
  <asp:ListItem>Item Two</asp:ListItem>
  <asp:ListItem>Item Three</asp:ListItem>
</asp:ListBox>
```

This is much the same as the way you add OPTION elements to a SELECT list in HTML.

The third is to use the `DataSource` property, where the items are taken from a collection (such as an array). For example:

```
Dim catList As new ArrayList

catList.add ("Item One")
catList.add ("Item Two")
catList.add ("Item Three")

DropDown.DataSource = catList
DropDown.DataBind()
```

Data binding is covered in more detail in the next chapter.

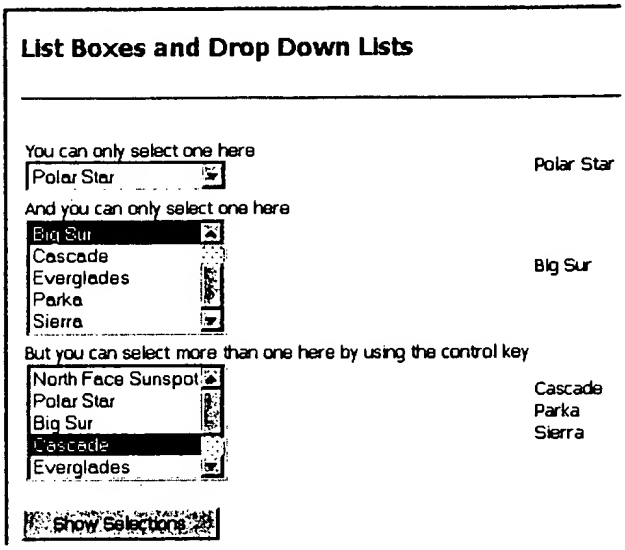
To examine the different selection modes, have a look at the code below:

```
<html>
<head>

<script language="vb" runat="server">

  Sub Page_Load(Sender As Object, E As EventArgs)

    If Not Page.IsPostBack Then

      ListsAdd ("North Face Sunspot")
      ListsAdd ("Polar Star")
      ListsAdd ("Big Sur")
      ListsAdd ("Cascade")
      ListsAdd ("Everglades")
      ListsAdd ("Parka")
      ListsAdd ("Sierra")

    End If

  End Sub
```

```
    Sub ListsAdd(item As String)

        DropDown.Items.Add(item)
        ListBoxSingle.Items.Add(item)
        ListBoxMultiple.Items.Add(item)

    End Sub

    Sub ShowSelectionsBtn_Click(Sender As Object, E As EventArgs)

      ' the drop down list only has one selection possible
      DropDownSelection.Text = DropDown.SelectedItem.Text

      ' as does the single selection mode list box
      ListBoxSingleSelection.Text = ListBoxSingle.SelectedItem.Text

      ' but the multi-selection could have more than one item
      Dim item As ListItem
      Dim sel  As String = ""

      For Each item In ListBoxMultiple.Items
        If item.Selected Then
          sel += item.Text + "<br/>"
        End If
      Next

      ListBoxMultipleSelection.Text = sel

    End Sub

</script>

<form runat="server">

  <table border="0">
    <tr>
      <td>You can only select one here<br/>
          <asp:DropDownList id="DropDown" runat="server" />
      </td>
      <td><asp:Label id="DropDownSelection" runat="server"/></td>
    </tr>
    <tr>
      <td>And you can only select one here<br/>
          <asp:ListBox id="ListBoxSingle" runat="server" />
      </td>
      <td><asp:Label id="ListBoxSingleSelection" runat="server"/></td>
    </tr>
    <tr>
      <td>But you can select more than one here by
          using the control key<br/>
          <asp:ListBox id="ListBoxMultiple" SelectionMode="Multiple"
               runat="server" />
      </td>
      <td><asp:Label id="ListBoxMultipleSelection" runat="server"/></td>
    </tr>
  </table>
```

```
    <br />

    <asp:Button id="ShowSelections" Text="Show Selections"
         onclick="ShowSelectionsBtn_Click" runat="server" />

</form>
</html>
```

The code here is quite simple. There are three selection boxes – one drop down list, one single selection list box and one multiple list box. The `Page_Load` event procedure simply loads all three with some values. In the event procedure for the button, we simply show the selected values. For the multiple selection mode we have to loop through each item in the list checking to see if it has been selected:



Images

The `Image` control is a replacement for the standard HTML image control, and really doesn't require much in the way of explanation. Its use is simple:

```
<asp:Image ImageURL="//localhost/advworks/images/dboots5.gif"
     AlternateText="A picture of some boots"
     runat="server" />
```

Containers

Container controls are controls designed to be parent controls, with other controls within them. You've already seen use of one of them – the `Label` control, which actually renders to a SPAN element.

The `Panel` renders to a DIV element, and is great for encompassing other controls, and is particularly useful when you need a group of controls to change their visibility. For example:

```
    <asp:Panel id="ControlsPanel" runat="server">

      Some arbitrary controls:<br/><br/>
      <asp:Label Text="A Label" runat="server"/><br/>
      <asp:TextBox Text="A TextBox" runat="server"/><br/>

    </asp:Panel>
```

Since the label and text box are contained within the panel, you could set the style of the panel such that it is hidden, and all child controls will also be hidden.

The `Table`, `TableRow` and `TableCell` map directly onto the TABLE, TR and TD elements in HTML, and are useful for building up tables that require server-side processing. For example:

```
<asp:Table id="ItemsTable" runat="server">
  <asp:TableRow>
    <asp:TableCell>R1 C1</asp:TableCell>
    <asp:TableCell>R1 C2</asp:TableCell>
    <asp:TableCell>R1 C3</asp:TableCell>
  </asp:TableRow>
  <asp:TableRow>
    <asp:TableCell>R2 C1</asp:TableCell>
    <asp:TableCell>R2 C2</asp:TableCell>
    <asp:TableCell>R3 C3</asp:TableCell>
  </asp:TableRow>
</asp:Table>
```

A far easier way to create tables is to use the DataGrid control, which creates rows and cells based on collections of data. This is covered in the next chapter.

Rich Controls

The controls described so far map fairly well onto their HTML equivalents, but just like the standard controls in Visual Basic, there's room for more. In VB we've seen a large market for advanced controls, with rich functionality, and although this has happened in the ASP market, it's not been to such an extent.

Since one of the tenets of ASP+ is its ease of use, you'll not be surprised to learn that an advanced set of controls comes with it. There are only two in the beta release, but we expect to see more from Microsoft in the final release, as well as a host from third parties.

AdRotator

The displaying of advertisements on Web pages is one of the few ways of making money on the Web. ASP originally shipped with an Ad Rotator, and this functionality has been encapsulated into a server-side control. It is extremely simple to use. First you must create the file of ads – this is an XML file, and takes the form:

```
<Advertisements>

  <Ad>
    <ImageUrl>/AdvWorks/images/AD_1.PNG</ImageUrl>
    <TargetUrl>http://www.astro-bikes.com</TargetUrl>
    <AlternateText>The Astro Mountain Bike Co</AlternateText>
    <Keyword>Biking</Keyword>
    <Impressions>80</Impressions>
  </Ad>

</Advertisements>
```

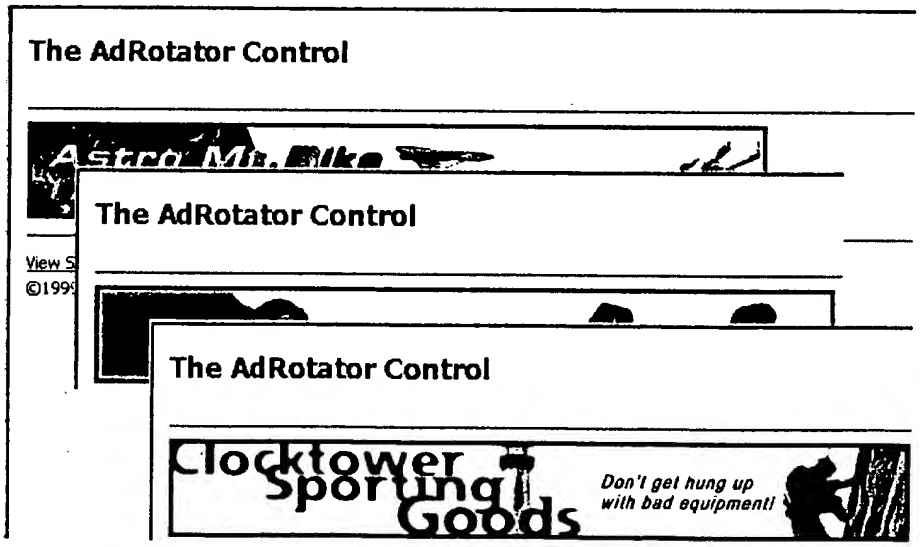
For each advertisement you require, just add an `Ad` element. Within this, you have the following:

- `ImageURL` is the URL of the image to display.
- `TargetURl` is the URL of the target Web site.
- `AlternateText` is the text to display when the mouse is over the image.
- `Keyword` is optional, and specifies a category for the advertisement. This can then be used in the `KeywordFilter` property of the component.
- `Impressions` indicates the relative weighting for the ad.

To include the rotator in your page, you just add the control:

```
<asp:adrotator AdvertisementFile="/advworks/xml/ads.xml"
     BorderColor="black" BorderWidth="1" runat="server"/>
```

This gives you an ad which rotates through the `AdvertisementFile`:



The XML file is also extensible, allowing you to add extra elements. For example:

```
<Advertisements>

  <Ad>
    <ImageUrl>/AdvWorks/images/AD_1.PNG</ImageUrl>
    <TargetUrl>http://www.astro-bikes.com</TargetUrl>
    <AlternateText>The Astro Mountain Bike Co</AlternateText>
    <Keyword>Biking</Keyword>
    <Impressions>80</Impressions>
    <Caption>Bikes that are out of this world</Caption>
  </Ad>

</Advertisements>
```

This extra element can then be made available in the event procedure for the component:

```
<asp:adrotator AdvertisementFile="/advworks/xml/ads.xml"
     BorderColor="black" BorderWidth="1"
     OnAdCreated="Ad_Created"
     runat="server"/>
<br/>
<asp:Label id="Caption" runat="server"/>

<script language="vb" runat="server">

  Sub Ad_Created(Source As Object, E As AdCreatedEventArgs)

    If e.AdProperties("Caption") <> "" Then
      AdCaption.Text = e.AdProperties("Caption")
    End If

  End Sub

</script>
```

Now, when an ad is created, the event procedure is run. This procedure checks to see if the property `Caption` has some text, and if so, it outputs it into the caption label. In this way you can easily customize your ads.

Calendar

The other control initially supplied is the `Calendar` control, which provides a rich calendar. Like the AdRotator it's extremely simple to use, although it does offer a great deal of customisation. Let's start with a simple example first:

```
<asp:Calendar runat="server"/>
```

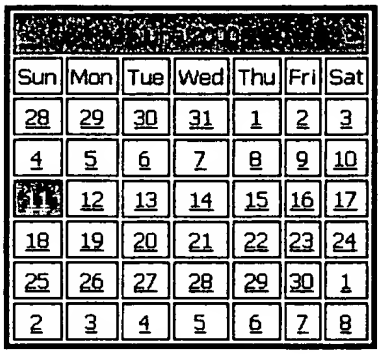
This gives the following simple output:

Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1
2	3	4	5	6	7	8

Pretty simple, but also pretty dull, so let's make it look better:

```
<asp:Calendar runat="server"
     BackColor="Beige" ForeColor="Brown"
     BorderWidth="3"
     BorderStyle="Solid" BorderColor="Black"
     CellSpacing="2" CellPadding="2"
     ShowGridLines="true"
/>
```

Now we have a different look:

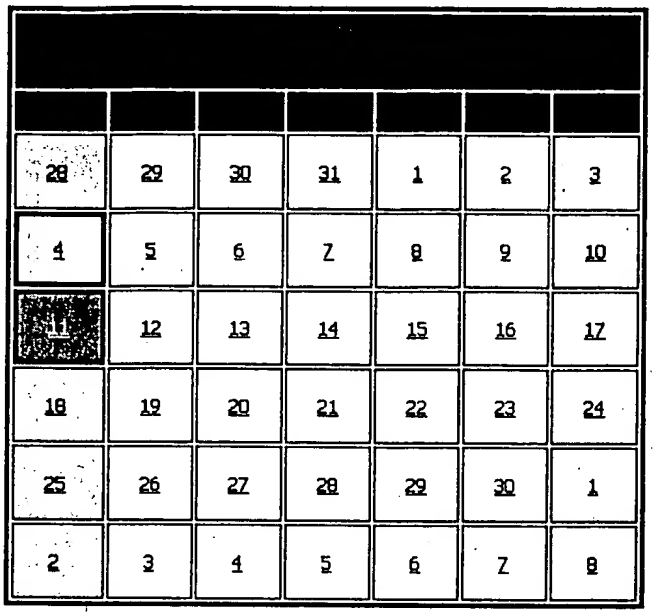


Sun	Mon	Tue	Wed	Thu	Fri	Sat
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1
2	3	4	5	6	7	8

Now, how about a bit more advanced formatting:

```
<ASP:Calendar CssClass="calstyle" runat="server"
     BackColor="Beige" ForeColor="Brown"
     BorderWidth="3"
     BorderStyle="Solid" BorderColor="Black"
     CellSpacing="2" CellPadding="2"
     ShowGridLines=true

     TitleStyle-BorderColor="darkolivegreen"
     TitleStyle-BorderWidth="3"
     TitleStyle-BackColor="olivedrab"
     TitleStyle-Height="50px"

     DayHeaderStyle-BorderColor="darkolivegreen"
     DayHeaderStyle-BorderWidth="3"
     DayHeaderStyle-BackColor="olivedrab"
     DayHeaderStyle-ForeColor="black"
     DayHeaderStyle-Height="20px"

     DayStyle-Width="50px"
     DayStyle-Height="50px"

     TodayDayStyle-BorderWidth="3"

     WeekEndDayStyle-BackColor="palegoldenrod"
     WeekEndDayStyle-Width="50px"
     WeekEndDayStyle-Height="50px"

     SelectedDayStyle-BorderColor="firebrick"
     SelectedDayStyle-BorderWidth="3"

     OtherMonthDayStyle-Width="50px"
     OtherMonthDayStyle-Height="50px"
/>
```

You can see that this has a rich set of styles available for customization, and gives the output overleaf:



28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	1
2	3	4	5	6	7	8

I won't go into detail explaining what the various styles are since it's fairly obvious.

Calendar Functionality

There are two events that are useful to respond to – when the date changes, and when the month changes. These can be hooked into the control in the following manner:

```
<asp:Calendar id="cal"
     OnSelectionChanged="cal_OnSelectionChanged"
     OnVisibleMonthChanged="cal_OnVisibleMonthChanged"
     runat="server"/>
```

Let's take the `OnSelectionChanged` event procedure first:

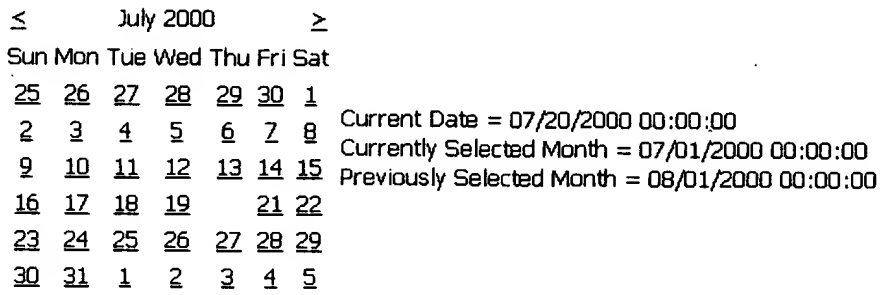
```
Sub cal_OnSelectionChanged(Source As Object, E As EventArgs)

  CurrentDate.Text = cal.SelectedDate.ToString()

End Sub
```

Here we simply set the `Text` property of a Label control to the currently `SelectedDate`. We have to convert it to a string first, because the selected date is a property of type `DateTime`. Using this event procedure you can synchronize other events, such as database queries dependent on the date.

For the month changing, the event procedure is as follows:

```
Sub cal_OnVisibleMonthChanged(Source As Object,
                              E As MonthChangedEventArgs)

  CurrentMonth.Text = E.NewDate.ToString()
  PreviousMonth.Text = E.PreviousDate.ToString()

End Sub
```

Notice that the second parameter to the event procedure is not just EventArgs, but contains more specific information. In fact, all it contains are two properties – `NewDate`, which is the new month, and `PreviousDate`, which is the previous month. Both of these dates are full dates, and are the first day of the month. This is easily seen when we run the sample:

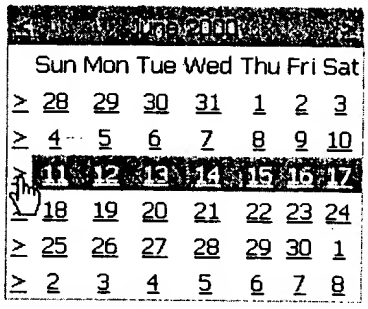


Here the current date shows the selected date, while the current and previous months show the first day of the appropriate month. If you only required the month names, then you could apply formatting to these dates.

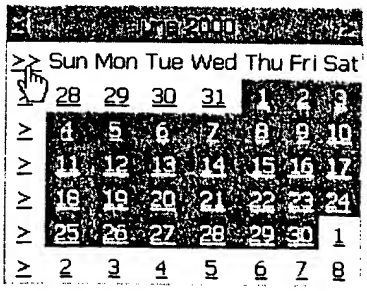
By default the calendar only allows selection of single dates. However, the `SelectionMode` property allows us to choose between the following:

- `None`, where no selection of dates is allowed.
- `Day`, where only the day can be selected.
- `DayWeek`, where a day or a whole week can be selected.
- `DayWeekMonth`, where a day, and whole week, or a whole month can be selected.

When in `DayWeek` mode, you get an extra column added to the beginning of each week, and selecting this selects the whole week:



When in `DayWeekMonth` mode you also get an extra selection at the beginning of the month:



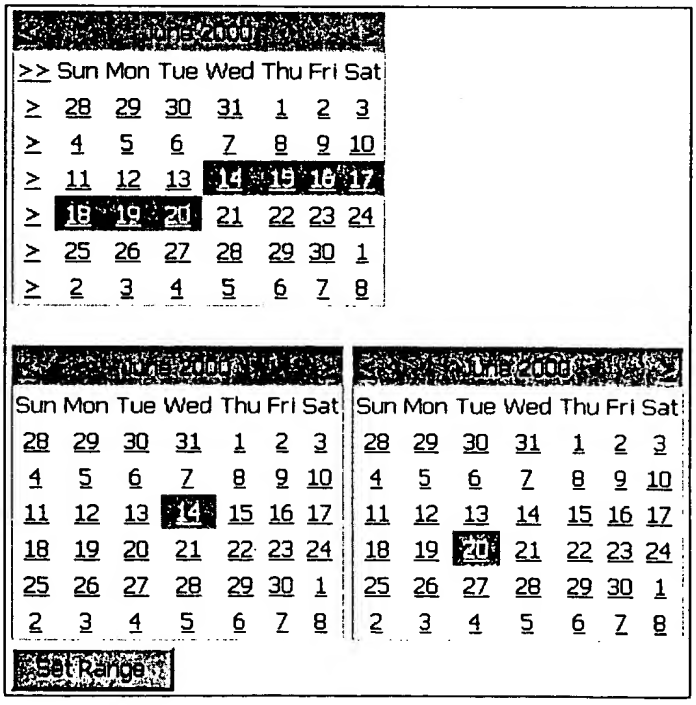
When you have multiple date selections like this, you can't use the `SelectedItem` property to determine which date has been selected, because more that one has. Instead you need to use the `SelectedItems` collection, which contains a DateTime for each date selected. So you could then change the selection event procedure to the following:

```
Sub cal_OnSelectionChanged(Source As Object, E As EventArgs)

  Dim sel As Integer

  sel = cal.SelectedDates.Count
  If sel = 1 Then
    CurrentDate.Text = cal.SelectedDate.ToString()
  Else
    CurrentDate.Text = cal.SelectedDates.Item(0) & " to " & _
                       cal.SelectedDates.Item(sel - 1)
  End If

End Sub
```

Here we check to see how many items have been selected. If it's only one, then we just display that one date. If more than one, then we show the first and the last in the range. You could also use a `For Each` construct to loop through all of the selected dates.

Date Ranges

You can also set ranges from code by using the `SelectRange` method of the calendar, which accepts two dates – that start and end of the range. So, you could easily add two more calendar controls to allow selection of the start and end dates, and then add a button to set the range on our existing calendar:



The code for the button looks like this:

```
Sub SetRangeBtn_Click(Source As Object, E As EVentArgs)

  cal.SelectedDates.SelectRange(calFrom.SelectedDate, calTo.SelectedDate)

End Sub
```

All in all, the calendar offers a great deal of functionality, with very little coding involved. That's the beauty of rich controls.

Other Rich Controls

The other set of rich controls that will be supplied with this beta are the Validation Controls, to aid in all types of form validation. These are covered in detail in Chapter 4.

The definitive list of other rich controls has yet to be decided, but it seems likely that more than these two will be supplied as standard. I know that the ASP+ team would like a Tree control, and some form of Menu control would be nice. But, since the product is still under construction, we'll have to wait and see. In the meantime, read Chapter 7 to see how to create your own controls.

Mobile Controls

For the final release of ASP+ there will be a set of controls for providing WML content to WAP devices. Since one of the ideas of rich controls is their ability to render differently depending upon the browser, you might think that this is an ideal situation – the controls could detect whether a WAP device was browsing them, and render content accordingly. However, the structure of WML means that input pages often have a different structure to standard HTML pages. For this reason, a separate set of controls will be produced.

At the time of writing no details of these have been released, but keep an eye on the supporting Web site and we'll keep you updated.

Custom Controls

One great advantage of the new ASP+ environment is that the controls you use aren't just limited to the supplied controls. As well as there being great potential for a third party control market, it's also possible to author controls yourself. This can be done in one of two ways – as custom controls or as Pagelets.

A custom control is one implemented in one of the supported languages, such as VB or C#. All of the supplied ASP+ controls are implemented in C#, and Microsoft is hoping the source for these will be available in the final release. This will not only help you understand how controls are written, but also help in authoring your own. In fact, the whole of ASP+ was written in C#, so there's no limit to what you can do. Writing your own controls is covered in Chapter 7.

Pagelets are custom controls that are implemented as ASP+ pages. This brings the ability to author controls within the reach of programmers who do not have deep familiarity with the control architecture. Implementation of Pagelets is covered in Chapter 4.

Summary

This chapter has been all about ASP+ Pages and Web Controls - the default set of controls supplied with ASP+. One of the things that should be obvious from reading this chapter is that not only do you get a set of functionally rich controls, but you can bring more consistency to your code. Under ASP when you posted back to the server, there was no way of separating the various parts of code. Under ASP+, with its event handling, it's extremely simple to logically split the code into the event procedures for selected controls.

It's now time to take your look at controls one step further, and delve into the world of handling data in ASP+ pages.

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.